ファミコンで
ここまでできる

ファミコン大作戦

ファミリーベーシック™
ジャンプ



**ファミリーベーシック
完全テクニックだ！**

グラフィックも
ミュージックも
自由自在に
プログラミング

地球防衛軍 編著
司令　谷岡康則

誠文堂新光社

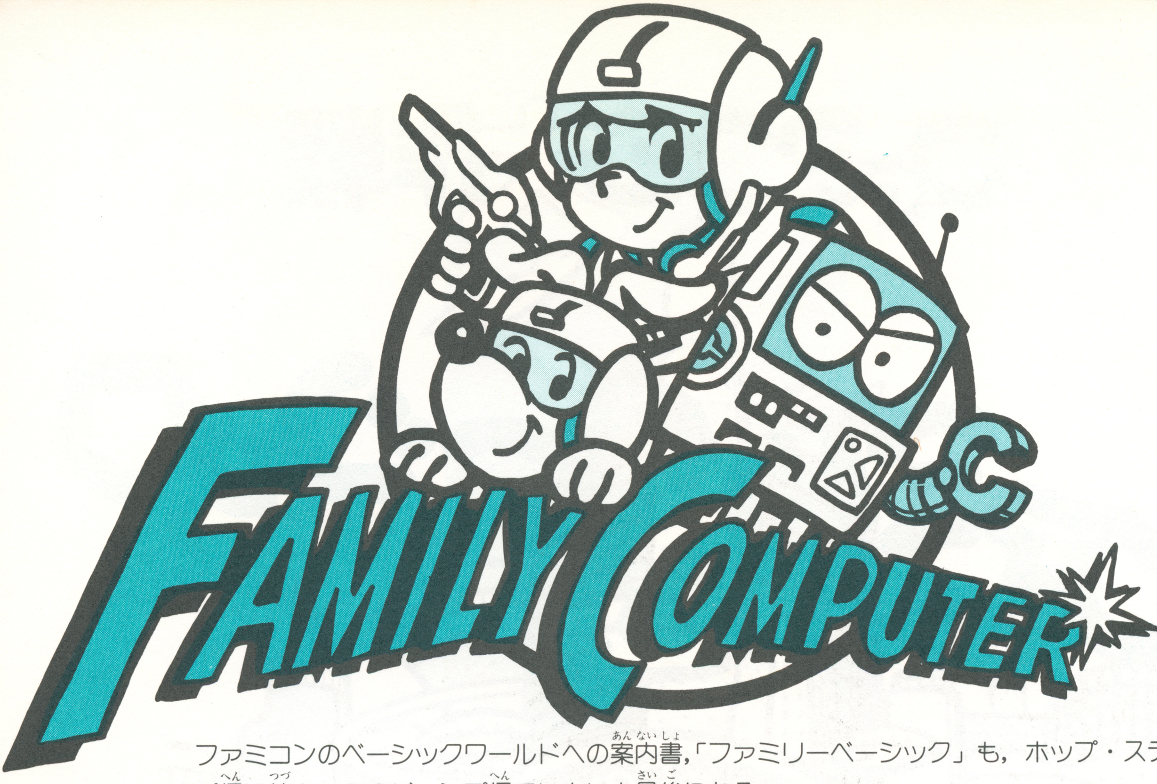
ファミコン大作戦　ファミリーベーシック™ジャンプ　誠文堂新光社

ファミコン大作戦

ファミリーベーシック



ジャンプ



ファミコンのベーシックワールドへの案内書,「ファミリーベーシック」も，ホップ・ステップ編に続き，このジャンプ編でいよいよ最後になる。

ホップ編では，何も知らなかった君もジャンプ編ではベーシックの通になってくれただろうか。このジャンプ編ではもう一歩，ベーシックの中身までふみこんでみた。

マニュアルにもくわしく書かれてないファミリーベーシックのカラーに対する命令や，音楽を演奏する命令も実例をあげながら紹介する。

Ver.3ベーシックの完全な説明や，少ないメモリーをいかに有効に使うかのテクニック。エラーメッセージとその対策にデバッグ法。

コンピューターの中身のCPVや，ベーシックの保存法・中間言語，そしてマシン語の世界のための第一歩，二進法・十六進法の説明もある。

パソコンで使ったリストアウトのプログラムまで公開したこのジャンプ編は，ファミコンの㊙情報のかたまりだ。

こうすれば，こうなる。という単なる説明書ではなく，なぜそうなるのか，なぜそうしなければいけないのか，私自身がコンピューターにはじめてふれた時にもった疑問を追求したのがこの本だ。

この三部作を読み直してくれた君は，もうファミリーベーシックについて友達に教えることができるだけの実力を持ったことを信じている。

ファミコンだけでなく，いつかパソコンにふれた時，そのパソコンのベーシックの働きが簡単に理解できるようになっていることを願っている。

一回読んだだけでは少し難しい所もあるかもしれない。そんな所は，何度も何度も読み返し，プログラムを入力してほしい。

入力したプログラムのここを変えればどうなるんだろう，と好奇心をもやしてほしい。

いつも，「なぜ」，「どうして」と疑問を持ってほしい。

コンピューター文明ともいえる技術が進歩している今の世界で，人間がコンピューターに支配されるようになってはおしまいだ。

コンピューターを利用し，使いこなすことが正しいコンピューターとのつきあい方だと信じている。

そしてそれができるのは，コンピューターという技術のかたまりを，ゲームに使っちゃえ，という発想をもてる若い君達の世代だけかもしれない。

少し固い話しになったけど，僕達にはコンピューターは友達だ。そしてコンピューターと仲良くなるには，コンピューターを理解しなきゃいけない。

この三部作が少しでも君とファミコンとのつきあいにプラスになってくれることを祈っている。

ファミリーベーシックを使った「ファミコンソフト集」も，001，002，003と，どんどん出版される。ゲーム作りのいろんなテクニックを公開して，君達と一緒にゲームを作ってしまう「ファミリーベーシック・ハイテクニック」や，ファミコンに対する君達からよせられたいろんな質問に答える「ファミリーベーシック・Q&A」も出版される。

これらの「ファミコン大作戦」シリーズが，いつも，君と，君のファミコンのそばにいて，一緒にコンピューターワールドに旅ができるのが僕達の夢だ。

地球防衛軍　谷岡康則

JUMP

もくじ

FAMILY BASIC JUMP

作戦指令008
音と色は変化をしつつ……

作戦指令009
これがVer.3ベーシックだ

●制作・企画　地球防衛軍　谷岡康則・岩田　裕・鈴木康夫・真藤美紀子・
簑藤　桂・北村　順・初瀬　龍・美崎靖之
●編集・企画　リポート社

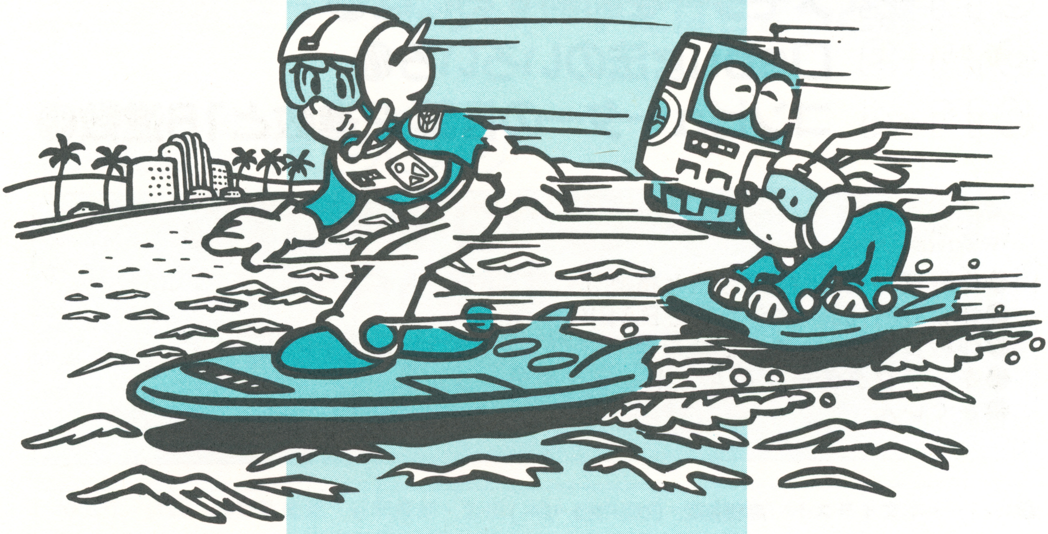
作戦指令010
ロールプレイングに挑戦

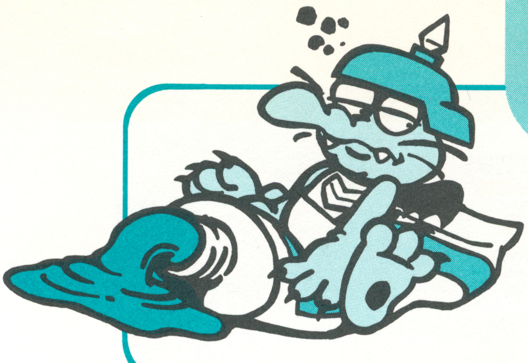
作戦指令011
ランダム・テクニック

巻末資料

●カバー・本文イラスト　高橋裕之（ファクトリー）

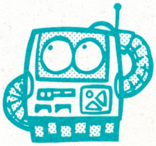
作戦指令
008
音と色は変化をしつつ…





008-1 パレットのきりかえは

52色のカラー



ファミコンは52色もの色が使える，というはずなのに，ステップ編で勉強した「COLOR」命令の3番目のパラメータ（命令のうしろについていて，その命令の効果をおよぼす場所や働を示す数字），つまり色の指定のところでは0から3までの4通りしか使えなかった。「DEF MOVE」や「DEF SPRITE」の色指定のパラメーターもやはり0から3までしか使えなかった。

たしかによく見ると，マリオやレディを表示させると，一つのキャラクターは3つの色で作られているらしいことはわかる。としても「3×4」の12色しか使えないじゃないか。ともかく，その12色を出してみよう。

●リスト1

```
10 CLS:SPRITE ON
20 FOR I=0 TO 3
30 DEF SPRITE 0,(0,1,0,0,0)=CHR$(0)+CHR$(1)+CHR$(2
)+CHR$(3)
40 SPRITE 0,120,140
50 PAUSE 100
60 NEXT
70 NEXT
```

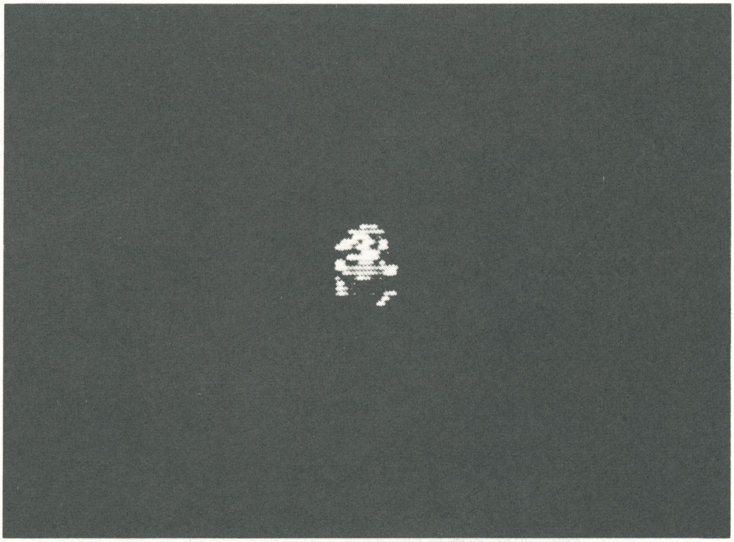
このプログラムを「RUN」するとマリオの色が4通りに変わっているのがわかるだろう。50行の「PAUSE」の後ろの数字を大きくすれば色が変わるのがゆっくりなるよ。さて上のプログラムに

●リスト2

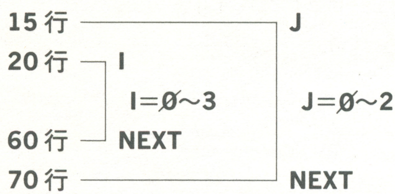
```
15 FOR J=Ø TO 2:CGSET 1,J


70 NEXT
```

この2行を追加してみよう。



この「FOR～NEXT」はわかるね。6Ø行の「NEXT」は，「I」をくり返すためのもので，7Ø行の「NEXT」は「J」のくり返し用なんだ。

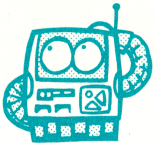


「J＝Ø」で「I＝Ø」から「I＝3」までの4回をくり返すと，次に，「J＝1」のままで，また「I＝Ø」から「I＝3」までの4回をく

り返し，今度は，「J＝2」として「I＝Ø」から「I＝3」の4回をくり返す。

このように「FOR～NEXT」ループを何重にもくり返すことを

多重ループ

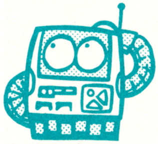


「**多重ループ**」（この場合は2重ループだね）と言ってプログラムではよく使うのは知ってるね。

だから，本当は

```
6Ø NEXT I
7Ø NEXT J
```

と書くと，もっとわかりやすいけど、ファミコンではエラーになってしまう。「NEXT」のあとには何も書いちゃいけないんだ。メモリーの余裕のある大きなコンピューターではリスト3のような書き方をすることが多いけど，ファミコンではちょっともったいないからだろう。（間をあける＝空白一字分ごとに，メモリーも減ってしまうね。）



CGSET

この「**CGSET**」というのはなんだろう。

●リスト3

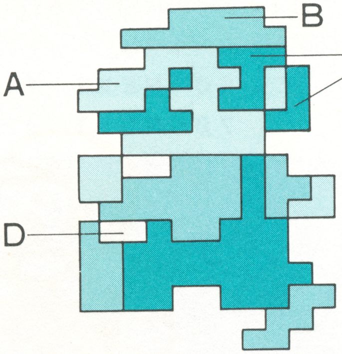
```
10 CLS:SPRITE ON
15   FOR J=0 TO 2:CGSET 1,J
20     FOR I=0 TO 3
30       DEF SPRITE 0,(0,1,0,0,0)=CHR$(0)+CHR$(1)+C
HR$(2)+CHR$(3)
40       SPRITE 0,120,140
50       PAUSE 100
60     NEXT
70   NEXT
```

ファミコンではたしかに52色のカラーが使える。しかし，普通のパソコンと違って，使われる絵はすべてファミコンの内部に登録されたものだけだ。パソコンだと，

LINE(X,Y)−(XX,YY),C　──→線を書く
PSET(X,Y),C　──→点を書く
PAINT(X,Y),C,C　──→色をぬる

のように，画面に自分の好きな絵を書いたり（上のような命令をグラフィック命令と呼ぶ）できるのだが，ファミコンにはそんな命令がない。

●図 1



ファミコンでは，３色で書かれたグラフィックキャラクタ（フォントともいう）を内蔵している。この３色をそれぞれ好きな色に塗り直せるようにしているのだ。

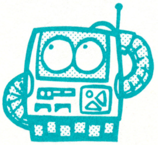
図１のマリオの略図を見てみよう。マリオはこのA，B，Cの３つの色と，透明の部分D（当然この部分は，背景色が見えるわけだ）で書かかれている。ということは

Aを青，Bを赤，Cを肌色　　にしたり

Aを緑，Bを白，Cを黄色　　にしたり

と色を変えることができるわけだ。

PALET



この３色をどういうふうな組み合わせにするか，というのは後で説明する「**PALET**」という命令で，君達が自由に決られるんだけど，ファミコン内部でも，標準的な組み合わせをもっている。それがこの本のカバーの折り返しにのっているカラーチャートだ。

スプライト用にパレットコード０，１，２，バックグラウンド用にパレットØ，１と，全部で５つのパターンがあるね。それぞれたてに４個，配色番号でØから３と書いてある。その一つ一つに三つの色があって，その下に数字がそれぞれ書いてあるね。

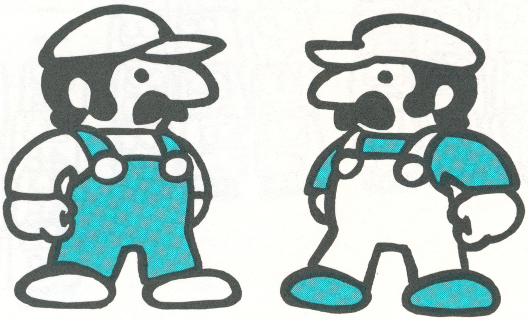
さっきの

```
CGSET 1,J
```

というのの，「J」は，このスプライト用のパレットコードからこの意味だったんだ。つまり

CGSET 1,Ø

とした後「SPRITE」命令の中の配色番号をØから3のどれかを指定することで，それぞれⒶからⒸの3色の組み合わせでキャラクタが表示されるわけだ。

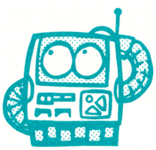


カンのいい君はもうわかったかな。

CGSET 1,Ø

の「1」はBG画面のグラフィックの色の組み合わせなんだ。ここで「1」を指定すると「バックグランド用」のパレットコード1を使うという意味だから，カラー番号のØから3はそれぞれⓠからⓣの組み合わせの3色を使うわけだ。

色の組み合わせ SETの交替



ただし，BG画面では常にこのⓠからⓣまでの組み合わせしか使えない。ただ，BG画面で書いたものを「VIEN」命令でキャラクタ画面へ転送した時

CGSET Ø

とすると，ⓠ→ⓜ，ⓡ→ⓝ，ⓢ→ⓞ，ⓣ→ⓟにそれぞれ色が変化するんだ。

このように

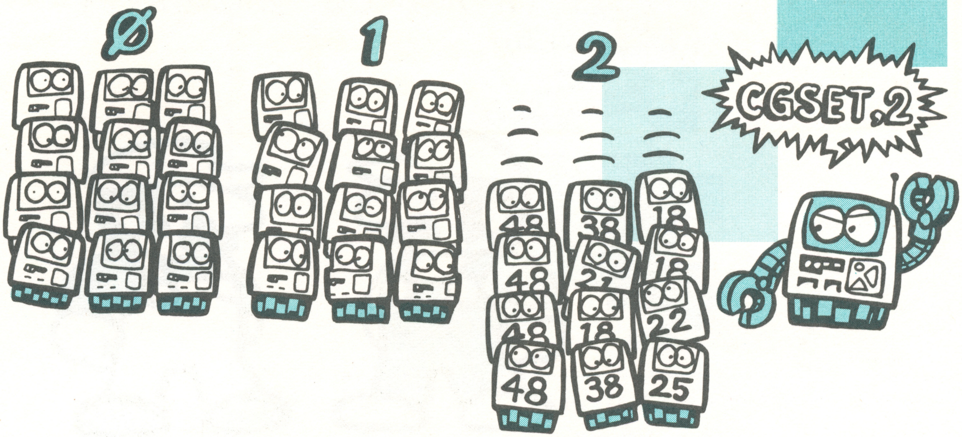
CGSET X,Y

は

CGSET X
CGSET ,Y

と，どちらか片方だけを指定することもできるんだ。ただし，スプライト面だけを指定する時は必ず「，」(カンマ)を入れようね。

Xが BG キャラクタでØか１，Yはスプライト用でØか１か２しか代入してはいけないのはわかるね。

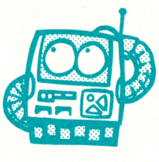


ところで，ホップ編やステップ編で勉強してくれた君なら，あれっ？と，首をひねるかもしれない。

ホップ編やステップ編では，この「CGSET」という命令を知らなかったから，当然使っていない。その時のマリオの色は図１のA⇨白，B⇨水色，C⇨青じゃなかったかな。この３色の組合わせは，「バックグラウンド用のパレットコード」１の配色番号Øと一緒じゃないかな。

実はその通り。電源を入れたあと，一度も「CGSET」という命令を使わなかったり，使ったあとでも，「Ctr.—D」をしたり，一度 BG 画面へ切り変えたあとベーシックへもどったりした時は，なぜか，BG キャラクタもスプライトも両方とも，バックグランド用パレットコード１」の組み合わせになってしまっている。

組み合わせ初期値



そして一回でも「CGSET」の命令を使うと，電源を切るか，「Ctr.—D」をするか，一度 BG 画面を呼びだすかしない限りは，スプライトで，この「BG 用パレットコード１」の色の組み合わせは使えなくなってしまうんだ。

なんとも理解に苦しんでしまう。変な話ではあるよね。

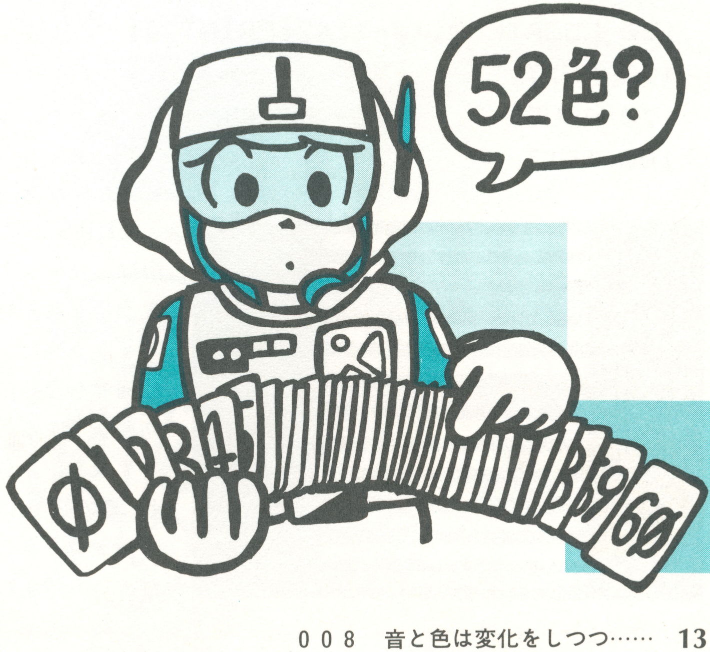


008-2
自分の好みにかえてしまう

前章では，使えなくなる。と書いたけど，実は方法はあることはあるんだ（何が？なんて聞く人は，もう一回この本を始めから読みなおし！）

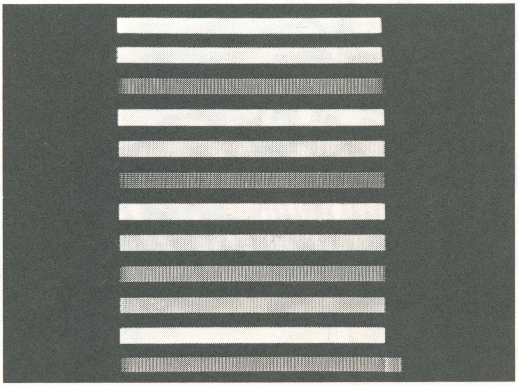
その前に，ファミコンで使える52色，というのはどんな色だろう。もうひとつ，マニュアル（ベーシックキットに付属している）をよく読んだ人なら疑問をもったはずだ。52色と言いながら色コード（それぞれの色にナンバーがついている）がØから6Øまで61個もあることだ。

ともかく次のリストを打ち込んでみよう。



●リスト4

```
10 CLS
20 FOR I=0 TO 3
30 FOR J=6 TO 20
40 FOR K=0 TO 4 STEP 2
50 COLOR J,I*6+K,I:LOCATE J,I*6+K:PRINT CHR$(K/2+2
53);
60 NEXT
70 NEXT
80 NEXT
90 PAUSE
100 FOR I=0 TO 240 STEP 12
110 FOR J=0 TO 3
120 PALETB J,15,J*3+I,J*3+I+1,J*3+I+2
130 NEXT
140 FOR J=I TO I+11
150 LOCATE 0,(J-1)*2:PRINT J;
160 NEXT
170 PAUSE
180 NEXT
```



CHR＄(253)，(254)，(255)というのは，マニュアルに「ヌリツブス」とあるようにそれぞれ一つの色でぬった真四角のキャラクタだ。

何もしない時（またはCtr.—Dの直後＝初期状態）この三つのキャラクタをプリントすると，それぞれ白，水色，青になったね。この3色は，「バックのグラウンド用パレットコード1」の配色番号∅の，3色だということに気がついた人はいただろう

か。さっそくこれを利用してみよう。

10行から80行は，この三つのキャラクタを横1列づつ，たて12段に表示している。この本のカバーおり返しのカラーチャートを見てみよう。

上から3番づつ，ⓠ，ⓡ，ⓢ，ⓣのそれぞれ3色になっているのがよくわかるよね。

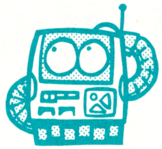
㊟ 50行のは，COLORでぬったあと，LOCATEでもう一度同じ場所へPRINTしている。これはVer.2ベーシックのバグのためで（ステップ参照），Ver.3の人なら普通に LOCATE, PRINT，COLORとしていい，この方が順序としてわかりやすい。

さて90行で「PAUSE」，何かキーを押すまでストップだ。

100行からは，新しい命令「PALET」を使って，カラーコードをØから251まで12個づつ画面に出している。それぞれの横線の左にでている数字が色のコード番号だ。

ちょっとまってよ，どんどんキーを押して行くと色コードが6Øをこしちゃった。マニュアルの説明と違うんじゃないか！

本当は何色?

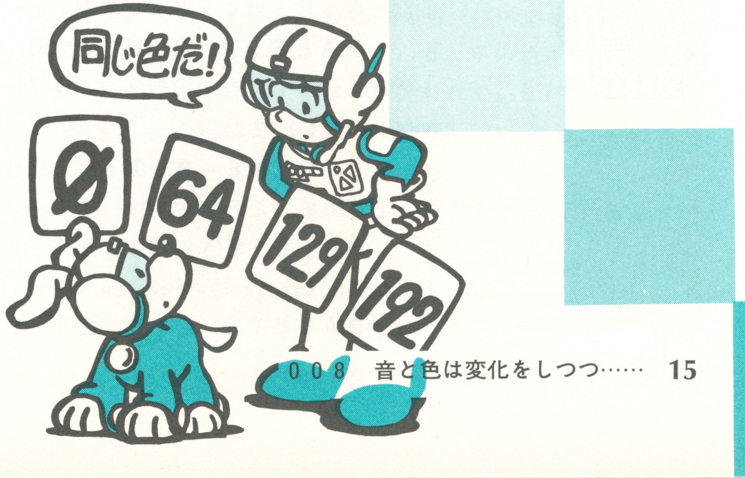


その通り。この色コードはØから255までの256通まで許されているんだよ。しかし，「Ø」と「64」と「128」と「192」は同じ色なんだ。つまり，**本当に意味のあるのは**「Ø」から「63」までの64通り。それより大きい色コードをXとすると

A＝（X MOD 64）

であらわせられるAという色コードと同じ色になる。

それでも，なぜ64通りの色が52色なんだろう。



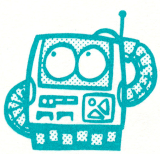
No.14，15，29，3Ø，31，46，47，62，63の8個は全部同じ黒じゃないかな。このように番号が違っても色が同じというのが何個かあるために，52色としているわけなんだ。

さて，120行にある「PALETB」が今回の新しい命令だ。この

●図2

色相（縦軸）／暗──明（横軸）

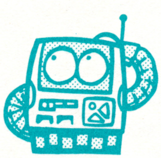
灰～白		Ø	16	32	48
有色	青系	1～4	17	33	49
	赤系	5～8			
	緑系	9～12			60
黒		13 14 15	31	47	61 62 63



「PALETB」こそ，カバーについているカラーチャートのⓐからⓣまでの3色の組み合わせをきめる命令なんだ。この命令には

PALETB

PALETS



PALETB

PALETS

の2つがある。「B」はBG画面やキャラクタ画面，「S」はスプライトだということはピンときたかな。使い方はほとんど同じ。

PALETB Ⓐ，Ⓑ，Ⓒ，Ⓓ，Ⓔ

とⒶからⒺまでの五つの数字（パラメータと言ったね）をつけなければいけない。また

LOCATE Ø，1 ⇨LOCATEØ，1

などと違って，この命令では

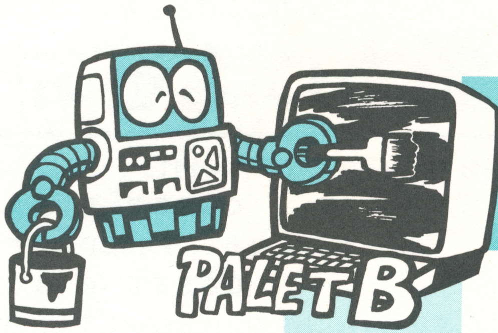
PALETBⒶ，Ⓑ，Ⓒ，Ⓓ，Ⓔ

と「PALETB」と数字Ⓐの間のスペース（空白）を省略したり，

PALETB ，，3，24，3

のように，変える予定のない数字を省略したりできないんだ。ただし，「PALET　B」のように「B」の手前はスペースをあけてもよい。

前章でのべたように，BGもスプライトも「CGSET」で指定した



カラーパレット・セットが常に一組づつしか画面上では使用できない。たとえば，電源を入れたり「Ctr—D」の直後では，BGもスプライトも，ⓠ，ⓡ，ⓢ，ⓣの四組の色の組み合わせを使うわけだ。

```
CGSET Ø,1
```

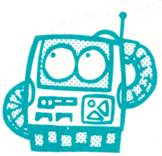
とすると，キャラクタ画面ではⓜ，ⓝ，ⓞ，ⓟの4組，スプライトでは，ⓔ，ⓕ，ⓖ，ⓗの4組だ。ここで，

```
PALETS
PALETB
```

とすると，ⓜ～ⓟとⓔ～ⓗの各組の色の組み合わせを変えることができる。ここで

```
PALETB Ø,13,5,6,7
```

とすると，BG⇨キャラクタの配色番号Ø（パレットパターンØだからⓜの3色44，21，7）の3色を5，6，7の色コードの組み合わせに変えるのだ。つまり，Ⓐは色が使われているパレットコードの中の配色番号（Ø～3だね），そしてⒸ，Ⓓ，Ⓔは，その3色を何にするかの色コードを書き込むわけだ。



背景色の変更

では，Ⓑは何だろう。ここは画面全体の背景色を変える命令だ。

```
PALETB Ø,22,5,6,7
```

とすると画面全体が赤くなるね。そのまま

```
PALETB 1,33,5,6,7
```

今度は水色等に変わるはず……でも変わらないね。このⒷに入れる数字はⒶ＝Ø（つまり配色番号Ø）の時に限って意味をもつんだ。

PALETB Ø,33,5,6,7

今度はちゃんと水色になったね。ⒶがØじゃない時はⒷの意味がないのならⒷを省略してもよさそうなもんだけど，やはり省略するとエラーになる。

「Ctr—D」をすると，またもとの色になる。ステップ編で説明したように「Ctr—D」には「CGSET　1，1」の役割もあったね。

さっきの例で

CGSET Ø,1

としたあと，PALET命令で色の組み合わせをいろいろ変えたあと

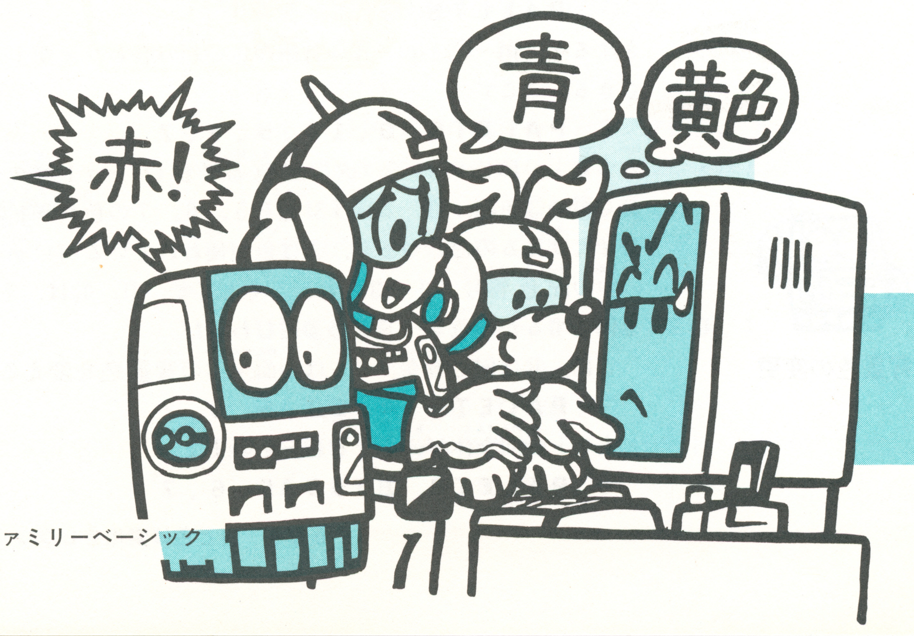
CGSET Ø,1

とすると，せっかく変えた組み合わせが，またもとの組み合わせにもどってしまっている。

PALET命令で変更しても，それが有効なのは，「Ctr—D」やCGSET命令を実行するまでの間でしかないんだ。

「PALETS」というのも命令によっていろいろ変わる組み合わせが，スプライト面にだけ有効というだけで，他の点では「PALETB」とまったく同じだ。いろいろためしてみよう。

なお，最初の背景面はカラーNo.13を使っているようだ。

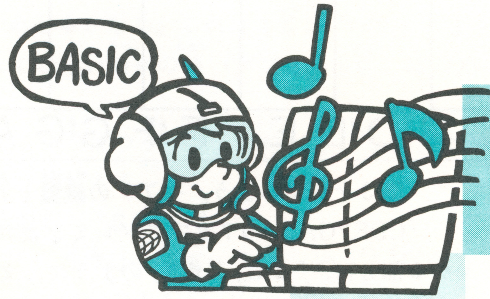




008-3 ミュージックボードとはちょっと違う

今度は音楽だ。というと，ホップ編でミュージックボードのことやったじゃないか，と恐るかな。

たしかに，ミュージックボードで音楽もやったけど，ベーシック中ではまったく違うやり方で音楽をならすんだ。めんどうくさいかもしれないけど，今度の演奏のしかたは，パソコンのベーシックにも共通する部分が多いから，ぜひがんばろう。



ミュージックボードと違ってベーシック上で音楽を演奏するには，当然，プログラム中でやらなければいけない。楽譜の〝オタマジャクシ〟にあたるものを文字列の形で記号化する。その文字列を「A$」とすると，

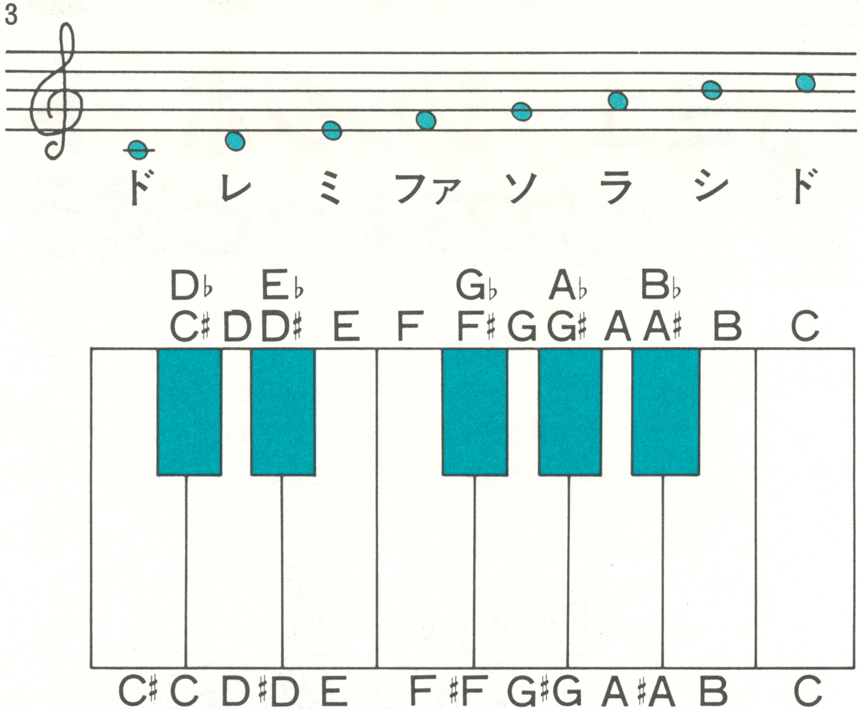
PLAY A$

で演奏してくれる。

非常に簡単に見える。問題は，譜面を文字列に置き変える，そのやり方だけである。

まず，「文字列は31文字以下」という長さの制限を覚えているね。そのことは常に頭に入れておこう。

●図3



♯と♭

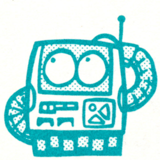


図3を見てみよう。ピアノの鍵盤と五線符だ。ベーシック上では「CDEFGAB」の７文字でドレミを表現する。ピアノの黒鍵にあたる部分は，「♯C」「♯F」のように，文字の前に「♯」(シャープ)をつける。

「C♯D」

というのは，「ドのシャープ，レ」ではなく，「ド，レのシャープ」という意味だから間違えないように。

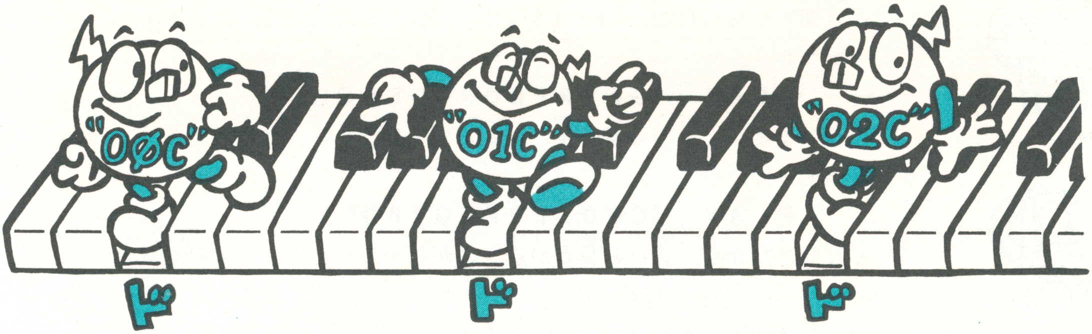
また，「♭」(フラット)記号は使わない。普通なら「E♭」のように使うが，これも「♯D」と書きあらわす。

これだけでは，１オクターブ(CからB)までしかできないので，「O」という記号を使う。

まず

```
PLAY "C"
```

としたあと

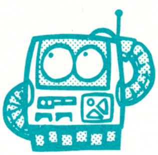


PLAY 〝OØCO1CO2CO3CO4CO5C〟

としてみよう。「C」は「ド」だ。最初は「ド」の一音だけを出すが，次の命令では低い音から高い音まで，6段階の「ド」を演奏したね。

最初に出した「ド」は，そのあとの六つの「ド」のうち，何番目の「ド」かわかるかなあ。これは4番目「O3C」，つまり，「O」でなにも指示していない最初の値（デフォルト値）は「O3」ということだ。

オクターブ



この「O3」は「**オクターブ**」の略だということはすぐわかるね。このままの状態でもう一度

PLAY 〝C〟

としてみよう。今度は一番高い「ド」，つまり「O5C」の音が出た。

この「O」は一度指定すると，次に指定するまでは，その値を変えない。これは，ファミコンだけでなく，ほとんどの演奏できるパソコンでもそうなんだが，その他の音楽用の命令も全部，一度決めたそれぞれの値は，次にかえると命令されるまで，ずっと保たれる約束だ。

PLAY 〝EGAGEGO4CCO3AAGECDE〟

これは「春の小川」の一節だね。「O4」で上の「ド」を出す。次の「ド」も上の「ド」だけど，「O」で一回指定すると変更しないからそのままでよい。そのかわり，次の「ラ」はもとのオクターブだから「O 3」と指定しなければいけない。

PLAY 〝GEEEFDDDCDEFGGG〟

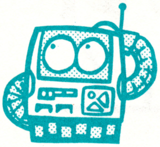
休符



これは「ちょうちょう」だけど少し変な感じだね。「休符」というのを音楽でならったね。「𝄽」とか「𝄾」だね。これは，「R」という文字をつかう。

```
PLAY "GEERFDDRCDEGGGR"
```

音の長さ



このままだと一つの音の長さが固定になっている。音符の長さはどうするのだろう。

```
PLAY "CØD1E3F5G7A9"
```

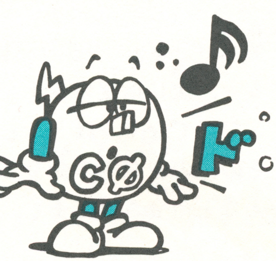
どうなったかな。音符の長さが違うのがわかるね。つまり音の高さの記号「A～G」や休符「R」のうしろに数字がつくことで，その音符の長さが決まるわけだ。この数字はØから9で，以下のようになる

Ø	𝅘𝅥𝅰	32分音符	5	♩	4分音符
1	𝅘𝅥𝅯	16分音符	6	♩.	符点4分音符
2	𝅘𝅥𝅯.	付点16分音符	7	𝅗𝅥	2分音符
3	♪	8分音符	8	𝅗𝅥.	付点2分音符
4	♪.	付点8分音符	9	𝅝	全音符

たとえば

```
PLAY "C5DE7FG"
```

としてみると，「D」は4分音符で「C」と同じ，「F」，「G」は2分音符で「E」と同じ長さなのがわかるだろうか。この，音符の長さも，一度指定すると，次に別の指定があるまで同じ長さになるんだ。



ところで，曲全体の速さ（テンポ）はどうだろう。曲の最初に「T」という字で指定できる。

```
PLAY "T3C"
PLAY "T6C"
```



2つの長さが違うのがわかるね。
音楽のテンポを表すのは，ふつう，1分間に演奏する4分音符の数で

♩＝112

のように書く。「T」のうしろには1から8までを使う。

T 1 ⇨	♩=448(𝅗𝅥=224)	T 5 ⇨	♩=9Ø
T 2 ⇨	♩=224(𝅗𝅥=112)	T 6 ⇨	♩=75
T 3 ⇨	♩=15Ø	T 7 ⇨	♩=64
T 4 ⇨	♩=112	T 8 ⇨	♩=56

が大体の速さだ。

テンポ

一曲の中で全体の**テンポ**を何回もかえるということはあまりないので，一番最初に設定しておくといいね。

この三つ，「T」と「O」と音符の長さで大体の曲は演奏できるが，他にもおもしろいものがいくつかある。

まず「V」だ。このVには二つの使い方がある

```
PLAY "MØV3C" ——①
PLAY "M1V3C" ——②
```

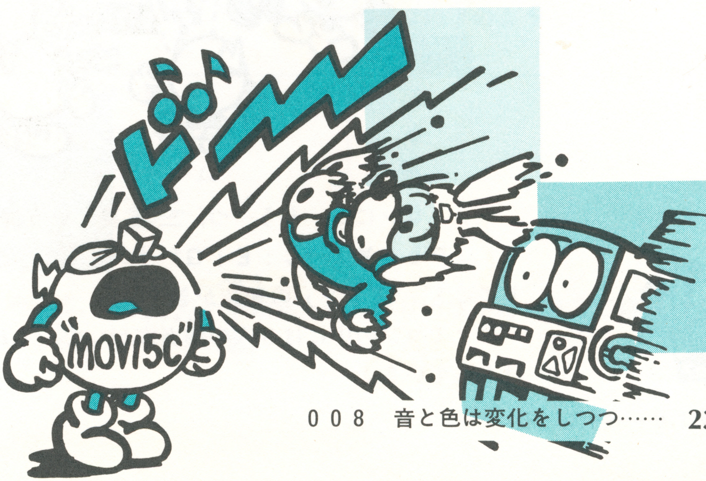
①を実行すると「ド」の音が小さくなったね。つまり，「MØV～」とすると音量（**ボリューム**）を指定することになる。「V」のうしろは1から15で，15が一番大きい音だ。

ボリューム

この「M」はエンベロープ（音に対する特殊効果）をかけるか，かけないかの指定で，「Ø」だとかけない，「1」だとかけるという意味だ。②のように，エンベロープをかけると，そのあとの「V」はエンベロープの長さの指定になる。この時ボリュームは15に自動的になってしまう。

音色

「Y」というのもある。うしろにはØから3の数字が入り，これを指定するとデューティ効果といって**音色**が変化する。ただし，これは必ず「MØ」か「M1」の命令のあとでないと有効にならない。



Øから3は，この効果をどれくらい加えるか，という数字で，以下のようになる。

Y Ø＝1 2．5％　　Y 2＝5 Ø％

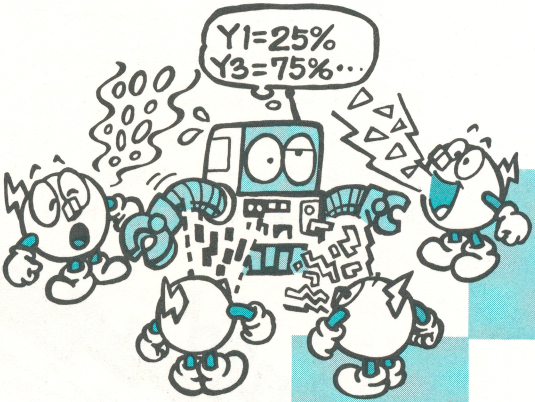
Y 1＝2 5　 ％　　Y 3＝7 5％

これらの効果を実験してみよう。

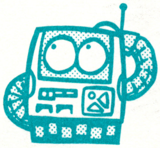
●リスト5

```
10 FOR I=0 TO 1
20 FOR J=0 TO 3
30 FOR K=0 TO 15
40 A$="M"+RIGHT$(STR$(I),1)+"Y"+RIGHT$(STR$(J),1)+
"V"+RIGHT$(STR$(K),LEN(STR$(K))-1)+"CR"
50 PRINT:PRINT "M=";I,"Y=";J,"V=";K
60 PLAY A$
70 NEXT:NEXT:NEXT
```

微妙な違いがわかるだろうか。



三重奏

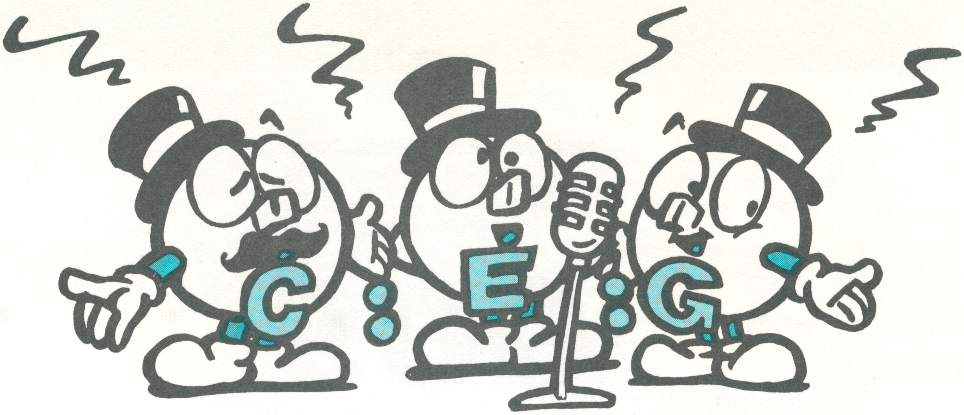


ファミコンには，もう一つ，三重奏という機能がある

PLAY　〝C：E：G〞

とすると，「ドミソ」の和音がきこえる。

ファミコンの中には3人の演奏家がいて，それぞれのパートをうけもっている。



PLAY　〝:C:E〞

PLAY　〝C::E〞

のようにすると，それぞれ，1番目の人，2番目の人は休み、という意味だ。

この三人に対しては，「M」や「Y」など，それぞれ別に指定しなければいけない。

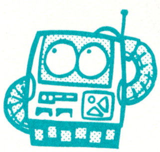
PLAY　〝MØY1C:C:M1V9Y3C〞

とすると，2番目の人は「M」や「Y」で何も指定しない場合の音をだす。

PLAY　〝C8CD:EF〞

とすると「D」の音は「C」と同じ長さだが，「E」や「F」は違う長さだってわかるね。長さやその他をいろいろ指定しても，「:」があると別になるわけだ。これらの各要素は以下のようになる。

初期値



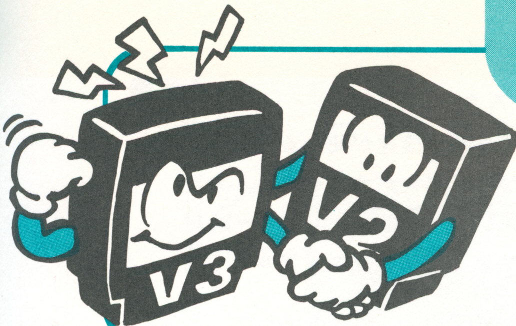
	初期値
T　テンポ	4
M　エンベロープ	Ø
V　ボリューム	15
O　オクターブ	3
音の長さ	5

そして，それぞれ一度指定した数値は再度指定するまで継続することを忘れないようにね。

作戦指令 009

これがVer.3ベーシックだ



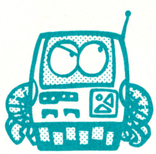


009-1 Ver.2との変更点

Ver.3ベーシックは基本的にVer.2ベーシックを強化拡張したものだ。だからVer.2ベーシックの命令はすべてそのまま使える。ただ例外がいくつかある。

```
CLEAR
SYSTEM
POKE
PEEK
```

SYSTEM



まず，「**SYSTEM**」だ。Ver.2ではベーシック画面でこの命令をダイレクトに打ち込むと，ベーシックとBG画面を選択すると画面になったね。ところが，Ver.3では電源ONするとこの選択画面が出ずに，すぐベーシック画面になる。BG画面にするには，新しい命令

```
BGTOOL
```

というのを使うと，そのままBG画面になる。つまり「SYSTEM」命令が不要になったのだ。働きとしては，「SYSTEM」も「BGTOOL」も同じようなものだから，同じ命令にしてほしかったんだけど，Ver.3で

```
SYSTEM
```

と命令すると，あとで説明する「BACK　UP」と同じ働きをしてしまう。

CLEAR



「**CLEAR**」命令には実は2つの働きがある。今まで使ってきた「CLEAR」は単に変数の値の記憶を消してしまう（**ステップ編**）だけだったが，もう一つ，こういう使い方がある。Ver.2の電源を入れたあと

```
CLEAR &H77ØØ
```

として

PRINT FRE

としてみると，「1982」にならないだろう。

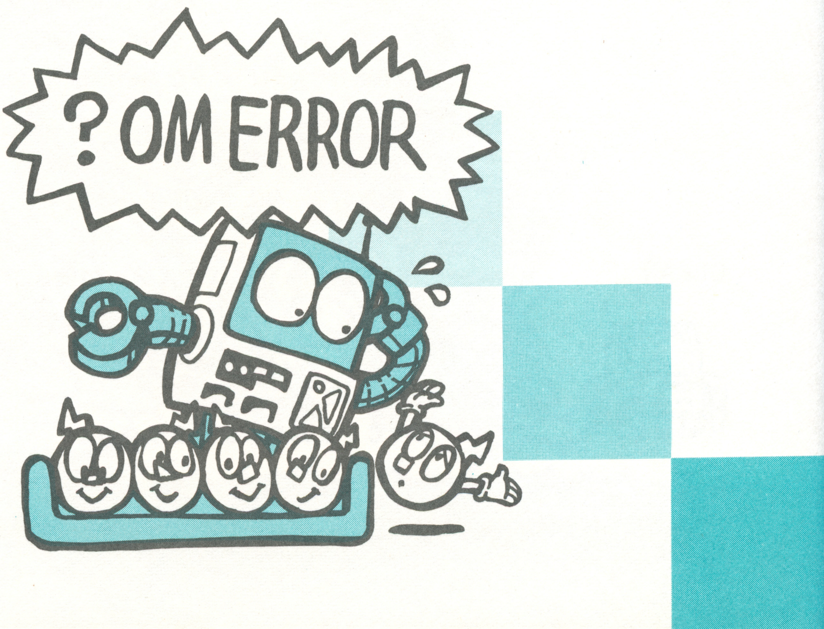
図１を見よう。コンピューターを使う時，僕達が使える部分は限られている。プログラムを書き込んだり，プログラムを実行した時に，中で使う変数の値を記録したりするための部分をユーザー領域というんだ。この場所がどこかは，ファミコンのマニュアルにも書いてある。

記憶する場所は１個分を１バイトと言うことは何度も話したけど，８ビットのコンピューターはその１バイトごとに番地をつけて管理している。Ø番地から65535番地までだ。これを16進法で（巻末の作戦資料を読んで）であらわすと&HØ番地から&HFFFF番地になる。

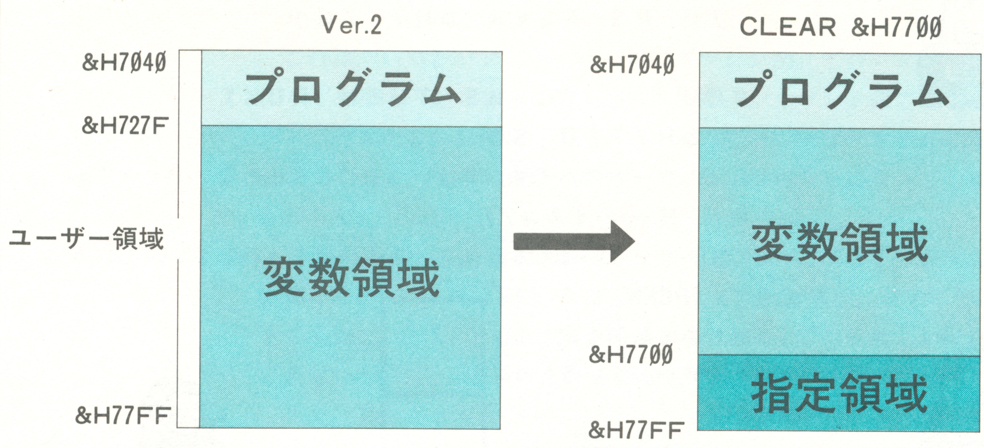
つまりVer.２では，このユーザー領域が&H7Ø4Ø（10進法の28736）から&H77FF（10進法の3Ø719）までというわけだ。

ファミコンでは図１のようにプログラムを&H7Ø4Øから書き始める。このプログラムが，たとえば&H727Fまでの長さだとしよう。プログラムを実行すると変数の値を記録する時，&H77FFから上の方（番地の若い方）へむかって変数の記録をはじめるわけだ。

どんどん記録をつづけていって&H727Fまで使わなければいけなくなると，もう書き込む所がないので，



●図1




```
? OM ERROR
```

つまり，メモリーが足りなくなる，というエラーがおきるんだ。

ところが

```
CLEAR &H77ØØ
```

と命令すると&H77ØØ（10進法の3Ø464）から&H77FF（10進法の3Ø719）は変数の値の記録には使えない，という宣言をすることになる。変数の記録をするのは&H76FF（3Ø463）より上のところを使う人だから，「FRE」の残りのバイト数が少なくなるのはあたりまえだ。（図2）なぜこんなことをするんだろう。

RUNすると前に記録した変数はすべて消えてしまう。ということは，何回もくり返してハイスコアを保存するためには

```
1ØØØ INPUT "REPLAY",A$
1Ø1Ø IF A$="Y" THEN RUN
```

なんてやってしまうと，ハイスコアの数値も消えてしまう。そこで，ゲームをリプレイする時は，GOTO文を使って処理しなければならない。

また，一度ゲームを中断したあと，もう一度「RUN」するとハイスコアが消えてしまっている。

そこで，たとえば，まずダイレクトで

POKE



```
POKE &H77F0,0
```

としたあと

```
10 HS=PEEK(&H77F0)
    ⌇
900 IF SC>HS THEN POKE
    &H77F0,SC
```

とすれば，ゲームのハイスコアはいつまでも変化しない (SC＝その回の得点，HS＝ハイスコア)。

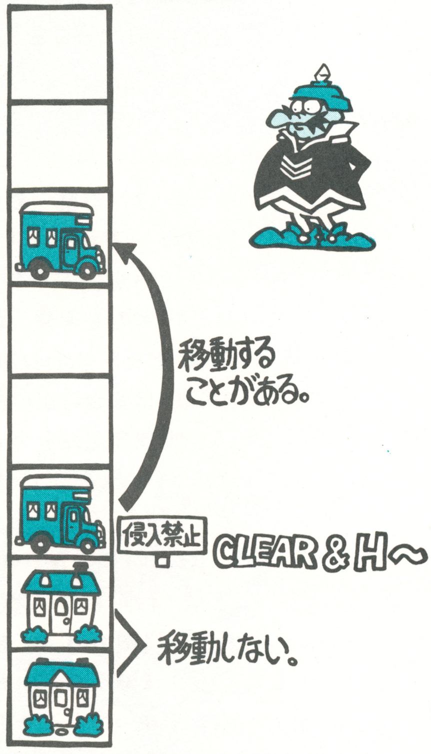
「POKE」というのはある番地に，ある数値を入れなさい。という命令で，「**PEEK**」はその番地にある数値を取り出しなさい。という命令だ。

PEEK



「A」という箱(変数)に「3」を入れるのと，どう違うだろう。

ベーシックで変数に値を入れるのは，トレーラーハウス（車で自由に動く家）に，似ている。コンピューターが「A」という箱の値を入れる。この「A」という箱をどの番地におくか（「A」という箱をつんだトレーラーをどこにとめるか）は，プログラムにより，そのつど，違うんだ。



「POKE」というのはそれに対し，絶対的な番地に対して数値を入力する，ということだ。ところが，前述したように，ある番地へPOKEしても，ベーシックのプログラムでその場所にトレーラーをとめたりすると，その番地の値がかわってしまう。そこで

```
CLEAR &H××××
```

のように，ここから先はトレーラー立入り禁止，という命令を与えておくと，安心なわけだ。

これは，いつかパソコンやなんかで「マシン語」でプログラムを作りだした時，絶対必要になってくる。せっかくマシン語で書いたプログラムをおいた所に，ベーシックの方でトレーラーをどんどんおいてくると，マシン語のプログラムがめちゃめちゃになってしまうからだ。

この「POKE」の2番目のパラメーターはØから255までの数字しか使えない。そして「PEEK」で返ってくる数値も当然，Øから255までの間だ。1バイトのおき場所にはØから255までの変数しか記録できないからだ。(巻末の作戦資料を見てほしい)

さて，Ver.2とVer.3の違いだ。この2つのメモリ・マップ(各番地の使い方を書いたもの) 図3を見てほしい。Ver.2では「ユーザー領域」が&H7Ø4Øから&H77FFまで，というのにくらべ，Ver.3では，&H6ØØØから&H6FFFまでにかわってくる。

この「CLEAR」文でうしろに番地がついたとき，当然その機能は，その番地がユーザー領域になければいけない。そこで

```
CLEAR &H76FF
```

はVer.2ではOKだがVer.3ではエラーになる。

また，「ROKE」文では「RAM」に書き込と書き込んだ値は保存されているが，「ROM」や未使用の所に書きこんで何の意味もない。

もし，これらの命令を使う時は，

```
1Ø INPUT "Ver.2 OR 3.",A
2Ø IF A<2 OR A>3 THEN 1Ø
3Ø B=Ø:IF A=2 THEN B=&H1Ø4Ø
```

とでもしたあと，「CLEAR」や「POKE」「PEEK」の目的番地を××××としたら

```
CLEAR &H××××+B
POKE &H××××+B,〜
PEEK (&H××××+B)
```

のようにしなければいけない。

●図3

(16進表現)

Ver. 2

アドレス	内容
&HØØØØ〜&HØ7FF	ワークRAM(ファミリーコンピュータ本体内)
&HØ8ØØ〜&H1FFF	未使用
&H2ØØØ〜&H5FFF	システムで使用
&H6ØØØ〜&H6FFF	未使用
&H7ØØØ〜&H7Ø3F	
〜&H77FF	ワークRAM(BASICカセット内)用 各種ボードデータまたはBASICフリーエリア
&H78ØØ〜&H7FFF	未使用
&H8ØØØ〜&HFFFF	プログラムROM

Ver. 3

アドレス	内容
&H6ØØØ〜&H6FFF	ユーザー(RAM)
&H7ØØØ〜&H7FFF	未使用域
&H8ØØØ〜&HCFFF	BASICインタ・プリタ
&HDØØØ〜&HD3FF	BG画面(マリオワールド用)
&HD4ØØ〜&HDBFD	GAME Ø
&HDBFE〜&HE681	GAME 1
&HE682〜&HF3Ø7	GAME 2
&HF3Ø8〜&HFFF9	GAME 3
&HFFFA〜&HFFFF	システム用



009-2

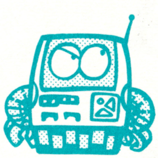
プログラム開発命令群

Ver. 3ではゲームのためではなく，プログラムを作る時に使う命令がいくつもできた。

①	AUTO DELETE RENUM	③	ERL ERR ERROR ON ERROR GOTO RESUME
②	TRON TROFF		
④	FIND INSTR	⑤	CLICK GAME

まず，①の3つの命令。これは行番号についての命令だ。ものぐさな人だと，いちいち行番号を入力するのがめんどうだな，と考える人がいるだろう。この「**AUTO**」という命令を使うと自動的にコンピューターが行番号をだしてくれるので，そのうしろに命令を書込むだけでいいのだ。

AUTO



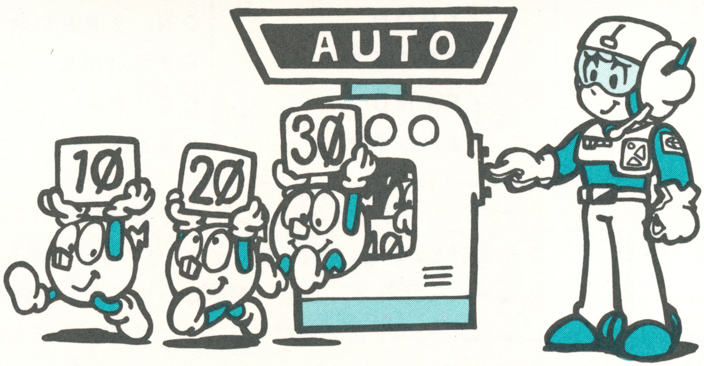
```
AUTO 1ØØ,1Ø
```

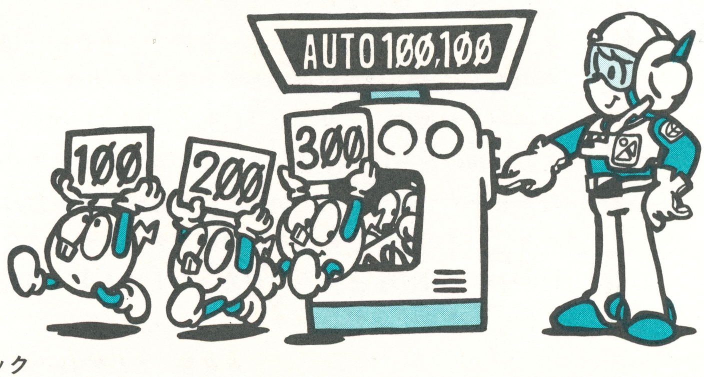
としてみよう。RETURNキーを押すとすぐに「1ØØ」と表示され，そのあとにカーソルが点滅している。何かてきとうに入力したあとリターンを押すと，今度は「11Ø」と出る。つまり，「AUTO」のあとの最初の数字は，行番号をいくつから始めるかの値，2つめはそのあといくつおきに出すか，という意味があったんだ。

もし，この時すでに何らかのプログラムがあった場合，たとえば110行が「END」というのがあったとすると「11Ø」の次に

```
11Ø END
```

```
OK
AUTO 100,10
OK
100 PRINT A
110 PRINT B
120 PRINT C
130 PRINT D
140 PRINT E
150 PRINT F
160 
```





と表示されカーソルは「E」の上で点滅する。変更を加えたくなければリターンだけ押せばいいし，変更したければその行をなおせばいい。

またこの時，たとえば105行が存在していても，表示されないで，最後にリストをとると残っている。

この「AUTO」モードをぬけでるにはSTOPキーを押せばいい。押さない限りはメモリーオーヴァーになるまで，いつまでも次の行番号が表示される。

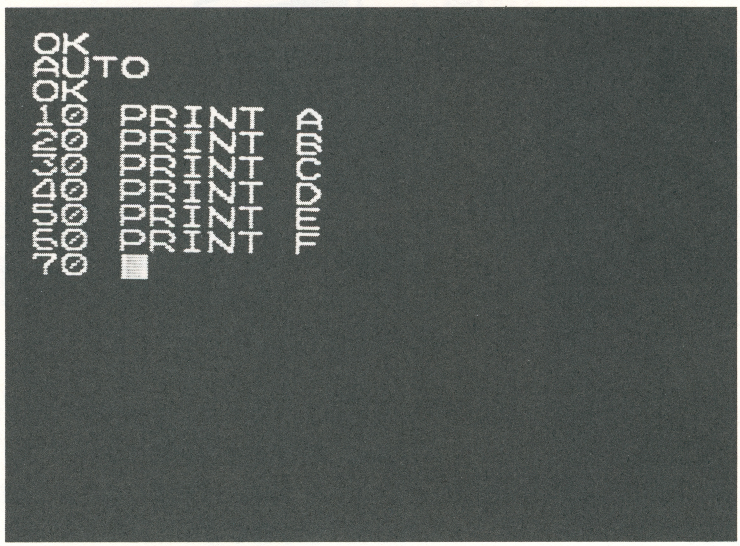
また，単に

AUTO

とだけすると，

AUTO 1Ø,1Ø

というのと同じになる。

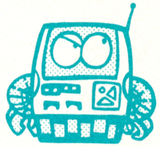


次に，いままでいらなくなった行を消すにはどうしていただろう。その行と同じ行番号だけを入力してRETURNキーを押していた。これは，同じ行番号で入力するのと，あとから入れた方がのこる。という性質を利用して，何の命令もない行を入力することで前の同じ行番号の命令を消していたわけだ。ところが，今度は，

DELETE 12Ø

とすると，120行が消えるようになった。これではたんに「120」＋リターンとする方が速いじゃないか。でも，この「**DELETE**」，「LIST」と同じ書き方ができるんだ。

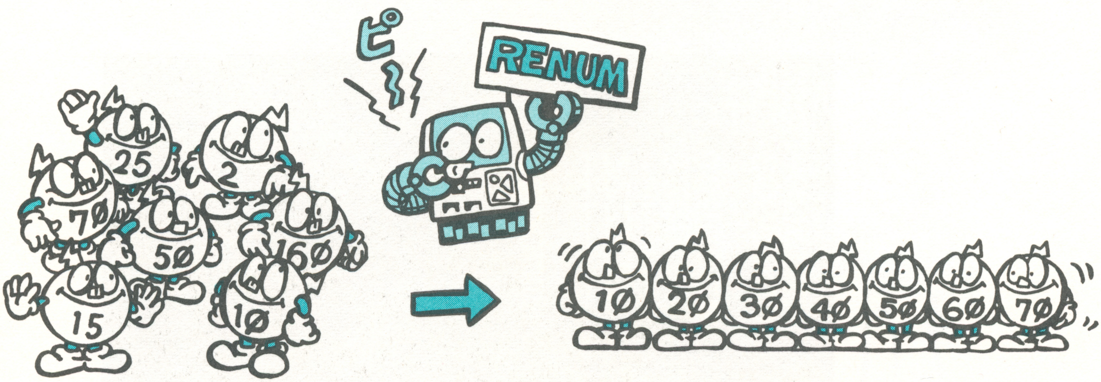
DELETE



DELETE 1ØØ－2ØØ	100行から200行まで
DELETE 1ØØ－	100行から最後まで
DELETE －2ØØ	最初から200行まで

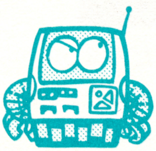
と，何行にもわたって一度に消せるんだ。(全部消すには「NEW」だったね)

ここでの「—」は「，」にかえてもかまわない。また行番号は実際にない行番でもかまわない。たとえば，10，20，30行があるプログラムで「DELETE　15—25」とすると20行が消えている。



プログラムを何回も修正していると，最初は10行から1Øおきに並んだプログラムも，20行と30行の間に22行や24行など半端な行番号がでてくる。これをきれいにそろえるのが「**RENUM**」だ

RENUM



RENUM Ⓐ,Ⓑ,Ⓒ

というように書く。Ⓐには新しくつける行番号，Ⓑには変更をはじめる行の行番号，Ⓒはその変更をいくつごとふやしていくかを書く。

今，10，12，20，40行がある。プログラムに

RENUM 1ØØ,12,2ØØ

とすると新しいプログラムは，1Ø，1ØØ，3ØØ，5ØØになっているのが

```
OK
10 PRINT A
12 PRINT B
20 PRINT C
40 PRINT D

RENUM 100,12,200
OK

LIST
10 PRINT A
100 PRINT B
300 PRINT C
500 PRINT D
OK
▒
```

```
OK
LIST
10 PRINT A
100 PRINT B
300 PRINT C
500 PRINT D
OK

RENUM
OK

LIST
10 PRINT A
20 PRINT B
30 PRINT C
40 PRINT D
OK
▒
```

わかる。

つまり12行からあとの各行を100行を頭に200行おきにする，ということだね。10行は当然そのままだ。

この時Ⓑを「12」じゃなく「11」としても結果は同じだ。

Ⓐを「5」にするとどうだろう。12行を「5」という行にすると前の10行と順番が入れ変わるね。こういう時はエラーになってしまう。順番が変わらない場合のみ，大丈夫というわけなんだ。

プログラム中で「GOTO」や「GOSUB」を使っている場合でも，それぞれの行番号はちゃんと新しい行番号に変わっているので安心してほしい。

ただし，この書き変えのためメモリーぎりぎりのプログラムだと，「RENUM」の作業ができないこともある。

RENUM

とだけすると

RENUM 1Ø,Ø,1Ø

と書くと同じ働きをする。プログラムの先頭から，最初を10行として，以後10行おきになおしてくれる。

Ver. 3のマニュアルのⒷのデフォルト値が1Øというのはまちがいなんだ。

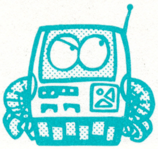
ところでファミコンの行番号，Øが使えるのは知ってたかな。パソコンでも0行は使えるものと使えないものとがあるんだ。

次に②の命令だ。

```
1Ø GOTO 3Ø
2Ø BEEP
3Ø GOTO 1Ø
```

こんなプログラムを作ってRUNしてほしいな。なにもおきないね。一度STOPキーを押してダイレクトで，

TRON



```
TRON
```

としたあとRUNしてほしい。

```
$1Ø $3Ø $1Ø $3Ø……
```

と次々に字が表示されているね。この「TRON」というのは現在実行中の行番号を画面に表示する働きをしろ，という命令なんだ。これを「トレースモード」といって，そこから抜け出すには

TROFF

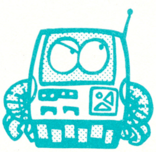


```
TROFF
```

という命令を使う。

プログラムを使ったあと，うまく動かない時に，どこに原因があるかを調べるために使うんだが，実験としてわかるように，次々と行番号が表示されるため，画面が完全に破壊されてしまうので，画面を使ったゲームのデバッグではあまり役に立たない。これはほとんどのパソコンがもっている「TRON」機能についても言えることで残念だ。

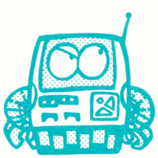
次の③の五つの命令も特殊なんだ。今までプログラム中はエラーが発生すると「?　××　ERROR」と表示されてプログラムの実行が止まってしまった。これらの命令は，もしエラーが出たらある行へ（エラー処理ルーチン）飛び，そこで何かの作業をして，もとへ戻って実行を続ける，という役割をもつ。



ERR と ERL

```
1Ø ON ERROR GOTO 1ØØØ
1ØØØ PRINT ERR,ERL
1Ø1Ø RESUME NEXT
```

こんな行を入れたプログラムを実行してみよう。



```
2Ø PRNT "A"
3Ø PRINT "TEST"
4Ø END
```

エラー処理

まず20行で「PRNT」が「PRNT」になっているからエラーになるね。今まではここで

```
?SN ERROR IN 2Ø
```

と表示されプログラムが止ってしまったが，「ON　ERROR　GOTO　1ØØØ」(エラーがあったら1000行へ行け）という命令のため，プログラムは1000行へ飛ぶ，そこで「エラーコードナンバー＝ERR」と「エラーをおこした行番号＝ERL」を表示して，1010行の，「エラーのことを忘れろ＝RESUME」，「エラーをおこした行の次の行へ行け＝NEXT」といわれて，30行へ飛び，「PRINT〝TEST〟」を実行する。

画面には

```
1          2Ø
TEST
```

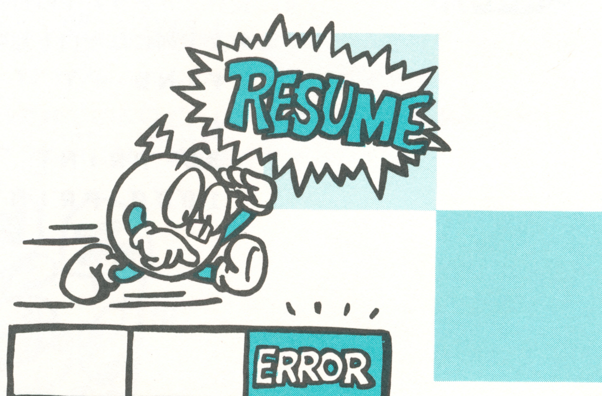
となるだろう。

これらの命令は一組になっている。「ON　ERROR　GOTO」でエラー処理ルーチンへ飛んだあと，メインルーチンに戻る時は必ず「**RESUME**」という命令をともなう。この時，単に「RESUM」とすると，エラーの発生した場所へ戻る。1010行を「RESUM」とだけすると，「　1　　　　　2Ø」をくり返かえしているだろう。

RESUME



「RESUME　××」とすると××行へ飛ぶ。「RESUME　NEXT」とすると，エラーを出した命令の次の命令へ飛ぶ。今のプログラムで



```
20 PRNT 〝A〞:PRINT 〝B〞
1010 RESUME NEXT
```

とすると，「TEST」と表示される前に「B」と表示されることでわかるだろう。

エラーコードナンバーとエラーの名前は，巻末のエラーコード表にある。

ERROR



「**ERROR**」というのは強制的にエラーを発生させる命令だ。

```
20 ERROR 5
```

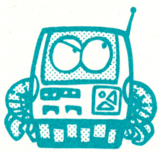
とすると，20行では命令のあやまりがないのに

```
5        20
TEST
```

となる。つまり20行で5番のエラーがおきている，とコンピューターに思い込ませたわけだ。

これらの命令は，パソコンで非常に大きなプログラムを動かしたり，何枚ものディスケットやテープを使うプログラムで，プログラム作者以外の人がコンピューターを操作する時に，ディスケットやテープの入れまちがいでプログラムを止めないためには非常に大事な命令だが，ファミコンではほとんど意味のない命令だ。

FIND



④のうち「**FIND**」はプログラム作成上，非常に助かる命令だ。

```
FIND 〝文字列〞
```

とすると，「〝」で囲まれた文字列がある行を画面に出してくれる。変数名を新しく作るときに，その変数がすでに使われたかを調べたり，長いプログラム中の特定の場所を探すのに便利だ。

この文字列には空白も意味をもつ。前記のプログラムに対して，

```
FIND 〝T 〞
```

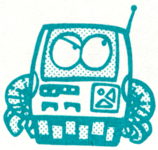
とすると

```
30 PRINT 〝TEST〞
1000 PRINT ERR,ERL
```

と表示される。いずれも下線部が"T　"にあたるわけだ。なお，この時，行番号は探す対象にはならない。(「FIND　"2Ø"としても20行は表示されない)

この機能はパソコンでも持ってない機種が多い。

INSTR



「INSTR」は，プログラムの中で使う。

```
10 A$="ABCDEFG"
20 PRINT INSTR(A$,"CD")
```

とすると「　3」と表示される。左側にもとの文字列，右側に探す文字を書く。

```
INSTR (A$,B$)
INSTR ("ABC","B")
```

のように書いてもOKだ。

Ver.3のマニュアルには

```
INSTR (3,A$,B$)
```

でA$の3番目の文字からB$を探してくれるとなっているが，我々のもっているVer.3カセットでは「？TM　ERROR」となって働いてくれない。これは使えないと思った方がいいよ。



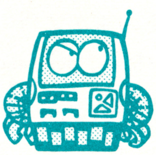
CLICK

最後は

```
CLICK ON
CLICK OFF
```

はそれぞれ，キーを押したときに出る「ブッ」という音を出したり，消したりする命令なんだ。

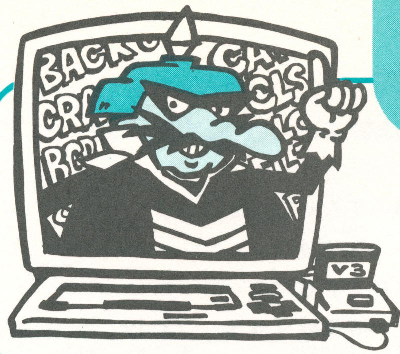
GAME



```
GAME
```

はROMカセットの中に登録されたおまけゲームを呼び出すための命令なんだが，四つあるこれらのゲームは，ファンクションキーの1から4に登録されているので，ファンクションキーを押せばいいだろう。プログラム中には一切関係がない。



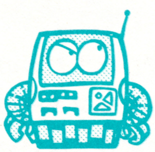


009-3 グラフィック命令の強化

Ver. 3ではBG画面とMOVE系の命令がたくさんふえた。

命令	
BGTOOL BGGET BGPUT LOADS SAVES BACKUP	①
SCREEN CLS FILTER	②
CAN CRASH VCT	③

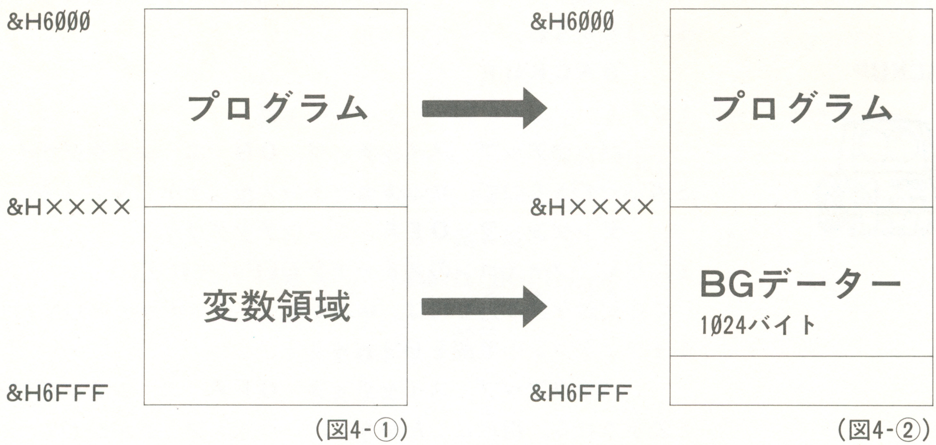
BGTOOL



「**BGTOOL**」は説明したように，Ver. 2の「SYSTEM」にあたる命令だが，「SYSTEM」がBGとベーシックを選ぶ画面に行く命令だったのにくらべ，これは一発でBG画面へ行く。そのぶん便利になったのだけど，なぜ命令語を変えるのかよくわからない。BG画面からベーシックへ戻るのはVer. 2と同じだ。①の残りの5つの命令は，バックアップやテープへSAVEする時の命令で，これまでにもおりにふれ説明してきたが，もう一度おさらいしよう。

まず，ロム・カセットに電池を入れるとバックアップ（プログラムを保存できる）のは知っていると思う。しかし，Ver. 2ではプログラムの保存だけで，BG画面は保存できなかった。Ver. 3も実はプログラムしか保存できないんだ。ところが，Ver. 3ではVer. 2の2倍もの記録する場所がある。そこで，プログラムがあまりにも大き

●図4

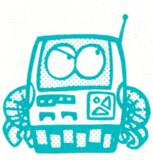


くなければ，プログラムのうしろ（普通は変数を記録する所，変数領域だったね）にBG画面のデータを移しちゃおう，と考えたわけだ。(図4）

バックアップしている時はプログラムは動いてないから変数領域も必要ないわけで，これはいいアイデアだ。BGデータは1Ø24バイトを必要とする。プログラムの大きさが3Ø63バイト以上だとこの機能は使えない。

```
CLEAR
PRINT ERE
```

として「1Ø24」以上の数字が表示されれば大丈夫だ。(「CLEAR」とするのは，プログラム実行中や終わりした直後は，変数に使った分が使用中とみなされるからだ。)



BGGET
BGPUT

BGデータをBG画面からユーザー領域のプログラムの後ろへ転送する命令が「**BGGET**」。逆にプログラムの後ろからBG画面へ転送する命令が「**BGPUT**」だ。

残りエリアが少ないのに「BGGET」をしようとすると「?OM ERROR」＝メモリーがたりない，「BGGET」していないのに

「BGPUT」しようとすると，「？NB　ERROR」＝BGデータがない，というエラーがでる。

さて，無事に「BGGET」できたら，図4―2の状態になっているのはわかるね。ここで，

BACKUP



BACKUP

とすると

バックアップ　スイッチ　ヲ　ON　ニ　シテクダサイ

と表示ができるので，ロムカセットのスイッチをONにする。

デンゲン　ヲ　OFF　ニ　シテクダサイ

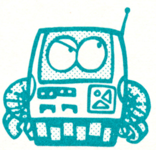
と出たら，ファミコンのスイッチをOFFにすればいい。

この記録を読み込むには，ロムカセットのスイッチをONにしたままファミコンの電源をONにすると，

バックアップ　スイッチ　ヲ　OFF　ニ　シテクダサイ

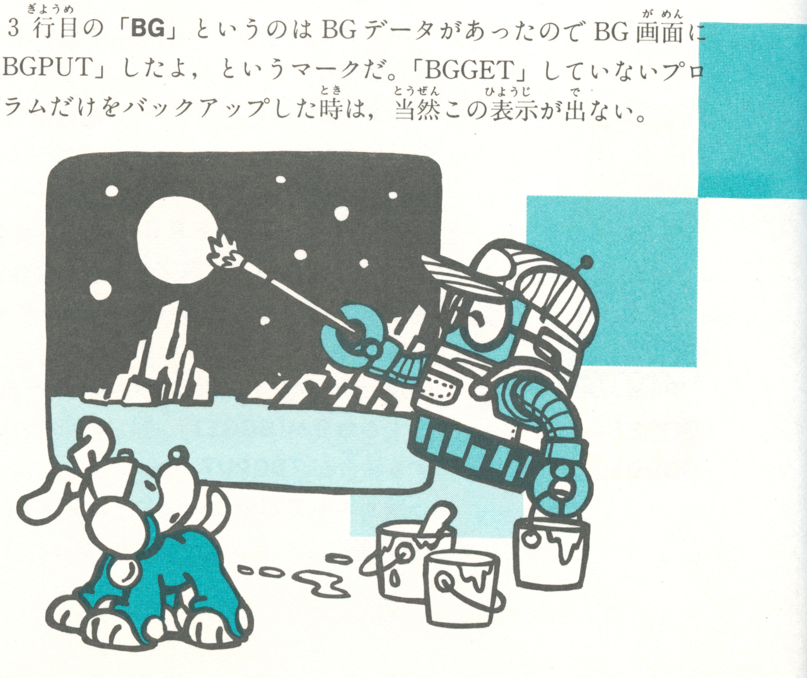
と表示される。指示にしたがうと，いつもと違う表示がでて，ベーシックがはじまる。

「BG」



```
BASIC HOT START
×××× BYTES FREE
「BG」
OK
```

3行目の「**BG**」というのはBGデータがあったのでBG画面に「BGPUT」したよ，というマークだ。「BGGET」していないプログラムだけをバックアップした時は，当然この表示が出ない。



注意しなければいけないのは，この時点で，自動的にコンピューターが「BGPUT」をしたので，このプログラムをもう一度バックアップするためには，もう一回「BGGET」しなければいけない。そうしないと，次にバックアップしたものはBGデータなしのプログラムだけになってしまうんだ。

図4－2をもう一度見てみよう。この状態で「SAVE」すると，テープには「&H××××+1Ø24」の所まで，つまりBGデータも一緒にテープにSAVEされる。このテープを「LOAD」すると，やはりBGデーターも一緒に読み込んでくれる。ただし，バックアップとは違って，自動的に「BGPUT」はしてくれない。自分で命令しなければならない。

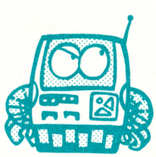
```
BGGET
SAVE
  ↓
LOAD
BGPUT
```

というわけだ。ここでも，一度「BGPUT」してしまうと，当然プログラム領域にはBGデータはないので，SAVEするには「BGGET」をしなければいけないし，プログラムを「RUN」する前にも必ず「BGPUT」をしなければならない。

㊟　ファミコンを使い込んでカセット差し込み口がゆるんでいる人は，バックアップ機能をつかわない方がいい。ロムカセットのスイッチを入れる時，カセットがゆれておかしくなる時がある。



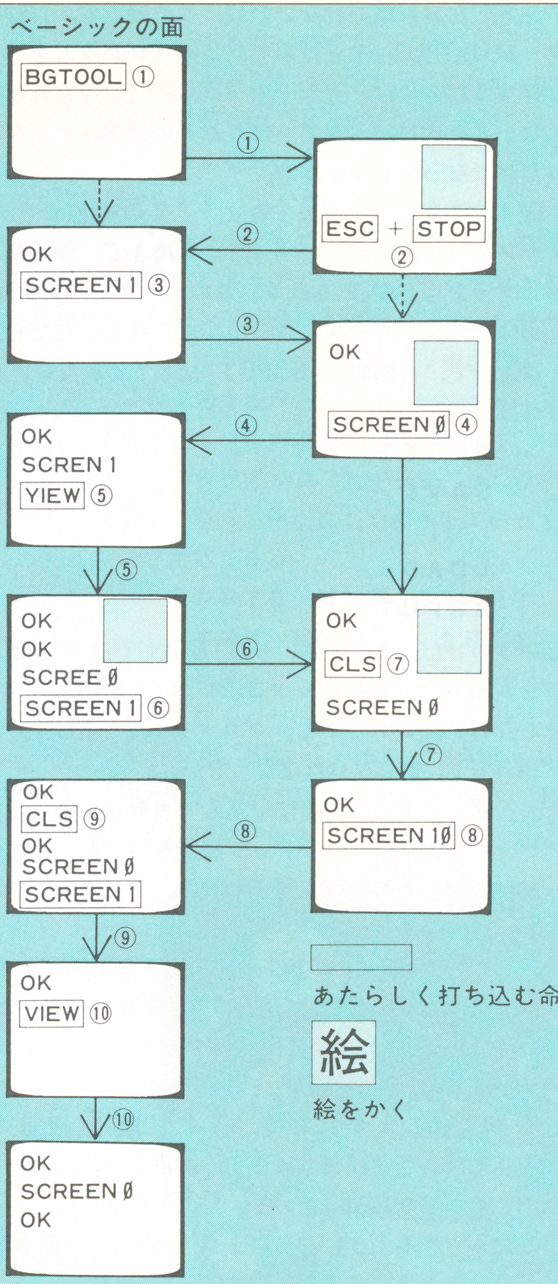
LOADS
SAVES



ベーシック画面上でBG画面データをLOAD，SAVEするのが「**LOADS**」，「**SAVES**」だ。この方法を使って作ったテープは，BG画面の「FILE」を選んで作ったテープとまったく同じものになる。

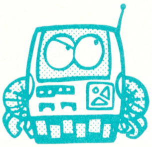
Ver.3では，BG画面が2つになった。とマニュアルにはある。残念ながら，そうではない。今までBG画面は「SYSTEM」や「BGTOOL」で呼び出されるBG画面書き込みモードでしか書いたり消したりできなかったのが，Ver.3ではベーシック上で書いたり消したりできるようになったということだ。BG画面書き込みモードで書けるのは前のようにBG画面だけだ。

●図 5



ベーシックの面
BGTOOL ①
①
ESC + STOP
②
②
OK
SCREEN 1 ③
③
OK
SCREEN 0 ④
④
OK
SCREN 1
YIEW ⑤
⑤
OK
OK
SCREE 0
SCREEN 1 ⑥
⑥
OK
CLS ⑦
SCREEN 0
⑦
OK
SCREEN 10 ⑧
⑧
OK
CLS ⑨
OK
SCREEN 0
SCREEN 1
⑨
OK
VIEW ⑩
⑩
OK
SCREEN 0
OK
あたらしく打ち込む命令
絵
絵をかく

スクリーンØ
スクリーン1



Ver.3では，BG画面のことを「**スクリーン1**」，ベーシック画面を「**スクリーンØ**」と呼ぶ。

ためしに，BG画面の右の方に何か絵を書いたあと，ベーシック画面にもひろう。

ここで，

```
SCREEN 1
```

としてみよう。今までの「VIEW」だとBG画面のデータが転送されてきたわけだから，今，入力した「SCREEN　1」という文字はBGデータと重ならない限りは残っているはずなのに，消えてしまっている。

```
SCREEN Ø
```

としてみよう。今度は前の「SCREEN　1」という文字が残っているけど，BGデータは消えているね。

図5を見てほしい。つまり

```
SCREEN Ø
```

で，「CLS」はベーシックの画面を消す。

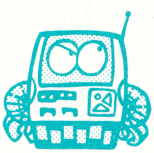
```
SCREEN 1
```

とすると画面はBGだから「CLS」をするとBG画面が消えてしまうのだ。

この「SCREEN」命令はパラメータを2つもつ。

```
SCREEN Ⓐ,Ⓑ
```

表示面と
アクティブ面



Ⓐはテレビに**表示する面**。Ⓑは「CLS」などの命令や入力した文字が有効な面（**アクティブ面**という）。

```
SCREEN Ø,1:CLS
```

としてみよう。カーソルが消えるが画面は消えない。この後，どんなキーを押しても変化がない。表示しているのはスクリーンØだがアクティブ面はスクリーン1だから，「CLS」もスクリーン1を消してしまったわけ。キーを入力してもそれは全部スクリーン1に表示されているので見えないわけだ。「Ctr—D」のあと

```
SCREEN 1
```

とすると，でたらめに押したキー入力がスクリーン1に表示されているのがわかる。

このように「SCREEN」のあとにパラメータが1個しかない時

は，表示面もアクティブ面もその数字の面にする，ということなんだ。

SCREEN 1 ⇒ SCREEN 1，1

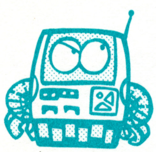
「CLS」以外に「LOCATE」や「SCR$」，「PRINT」等，画面に対する命令はすべてこのアクティブ面に対しておこなわれる。

「CLS」も少し変わった。単に「CLS」とするとアクティブ面を消すが，「CLS　Ø」や「CLS　1」とすると，現在のアクティブ面はかかわりなく，指定したスクリーンを消去する。

残念なのは，２つの画面を有効に使えるようにしたにもかかわらず，「VIEW」はあいかわらず一方通行でスクリーン１をØへ転送する役目しかない。だから，一度でもBG画面を「CLS」してしまうと，BG画面をもとに戻す方法はほとんどない。(非常にめんどうくさく時間がかかる)

よほど注意してプログラミングしないと，せっかくの機能も宝のもちぐされだ。

FILTER



FILTER Ⓐ

とすると，スクリーンØ，１とも全面に淡い色がつく。ⒶはØから７の数字が入る。

Ø――	無色	2――	緑
1――	赤	3――	黄



4―― 青　　　　6―― 空色
5―― マゼンダ　　7―― 白

もとに戻すには

FILTER Ø

とするか「Ctr―D」だ。「Ctr―D」はグラフィック関係のすべての値をもとの値（デフォルト）に戻す役割をもっている。

③の三つの命令は，MOVE系の命令強化だ。

VCT



まず「**VCT**」・「DEF　MOVE」で定義したキャラクタがどっちの方向に移動しているかを教えてくれる。

DEF　MOVE（W）＝SPRITE～

A＝VCT（W）

のように書くと，「A」の中に「W」の移動方向が入る。「W」は当然Øから7までの数字が入り，Aは

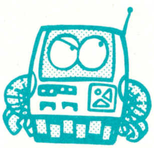
```
 8  1  2
  ↖ ↑ ↗
 7←Ø→3
  ↙ ↓ ↘
 6  5  4
```

の数字が入る。

残る二つは，Ver. 3になっての最大の収穫である便利な命令なんだ。

Ver. 2では「DEF　MOVE」で定義したキャラクターを移動途中で消して，しかも次に新しい場所から動かす。つまり，表示と座標位置を未定義にするには，「ERA」と「POSITION」が必要だったね。ところが，今度は

CAN 3



CAN

のように簡単にこれができるようになった。

「**CAN**」のうしろには

CAN 3,6,7



CRASH

のように1度に何個ものキャラクタを指定できるのは，「ERA」や「CUT」と同じだ。

最後に「**CRASH**」。Ver. 2では，たとえば「レーザー」のキャラクタと敵のキャラクタがぶつかったかを判定するには大変だった。



両方のＸ座標・Ｙ座標のそれぞれがある数値以内かどうかを，長い「IF」文で判定していた。ところが，この「CRASH」では一発で，できるようになった。

「DEF　MOVE」で決めているキャラクタナンバーが，敵は「Ø」，味方は「１」，敵のレーザーが「２」，味方のレーザーが「３」としよう。

IF　CRASH（３）＝Ø　THEN　　敵を破壊
IF　CRASH（２）＝1　THEN　　やられた

これだけでいいんだ。つまり，

CRASH（W）

は，「W」番のキャラクタが他のスプライトと重なっているかを数字で教えてくれるんだ。「－１」だと重なっていない，まちがって「W」を「DEF　MOVE」で定義してなければ「－２」，もし重なっていたら，「Ø」から「１」の数字で返してくるんだ。

ただし，この数値は重なっているスプライトの内，一番小さなスプライト番号で帰ってくるので，特に３個以上が重なる可能性がある時は，スプライトの番号をうまくつけないとうまくいかないのは，わかるね。

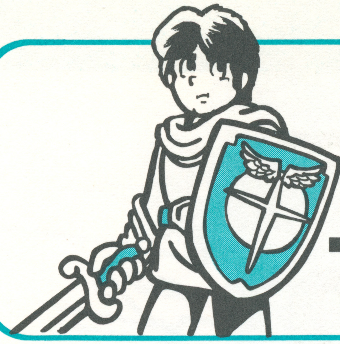
この機能だけはVer.２でも，ぜひほしかった。と思わないかい？

作戦指令 010

ロールプレイングに挑戦



0 1 0 - 1



ロールプレイングとは

今，パソコン界で一番流行しているのがロールプレイングだ。ホップ・ステップとゲームを作ってきたけど，いよいよ今度はロールプレイングに挑戦だ。

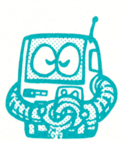
でも，ロールプレイングってなんだろう。

リアルタイムゲームではメインキャラクターは，どんなに長くゲームが続いていても変化はないね。たとえば「ロードランナー」も，面数が進んでも，力や走るスピードがあがるわけじゃない。

「役割を演じる」というのがロールプレイングの意味だ。このゲームでは，プレイヤーはゲームの主人公になる。そして主人公が行なう冒険は，プレイヤーの冒険になるわけだ。

みんなも学年があがっていくとだんだん授業が難かしくなって行くね。小学校1年の時，お兄さんやお姉さんの教科書を見ても何が書いてあるか全然意味がわからなかっただろう。ところが，一年一年，学年が上がってくるといつのまにか，わかるようになるだろう。つまり，君は成長してきたわけだ。

成長する主人公



ロールプレイングの**主人公**もみんなと同じに，成長してくるんだ。敵をやっつけたら経験値が上がる。良い武器を手に入れたら攻撃力が上がる。でも戦闘をすると体力が減るし，食料がないとうえ死にてしまう。休息をとったり食事をして体力を回復するしかない。多くのロールプレイングでは経験値が上がると魔法が使えるし，だんだん高度な魔法が使えるようになる。そのかわり，敵も強くなってくる。

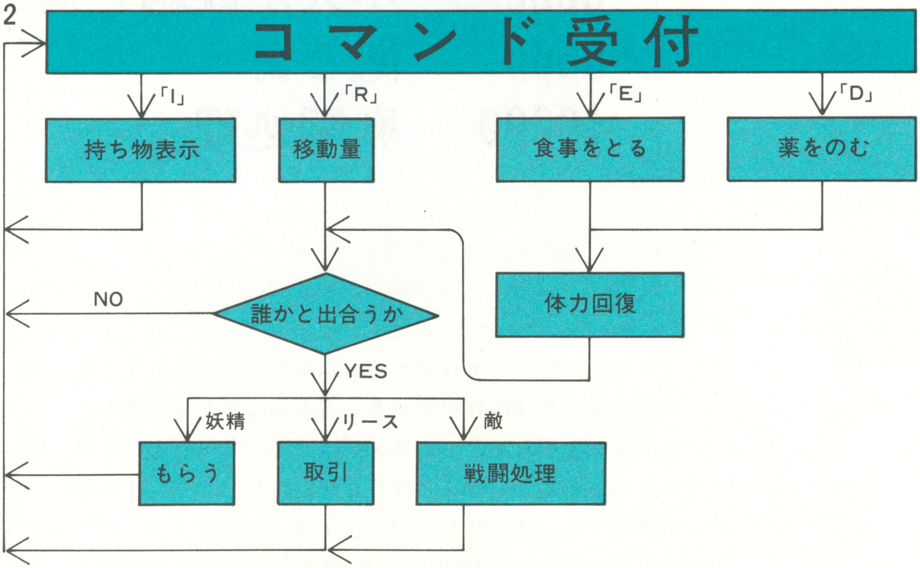
主人公は最後の目的を達成するためには，少しづつ，自分の力を強くしなければいけないんだ。

最高のロールプレイングはゲームを終了するには何ヶ月もかか

●図 1

A	(Ø)	主人公の経験値	= Ø
	(1)	主人公の体力	=1Ø5
	(2)	主人公の戦闘力	=1Ø5
	(3)	主人公の体力上限値	=1Ø5
	(4)	主人公がもっている食料の数	=1Ø5
	(5)	主人公がもっている武器のレベル	= 1
	(6)	主人公がもっている薬	= Ø
	(7)	主人公の位置	= 11
O	(	王冠のある階	
B	(Ø,Ø)	兵士の基礎体力	= 1Ø
	(Ø,1)	兵士の基礎戦闘力	= 1Ø
	(1,Ø)	ゴーレム基礎体力	= 15
	(1,1)	ゴーレム基礎戦闘力	= 15
	(2,Ø)	ドラゴン基礎体力	= 2Ø
	(2,1)	ドラゴン基礎戦闘力	= 2Ø

●図 2



●図 3

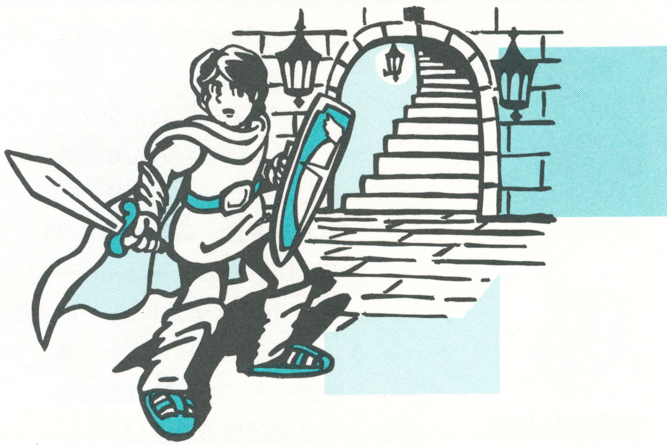
10	初期値
100	表示
200	コマンド処理
1000	「I」の処理
2000	「E」の処理
3000	「D」の処理
4000	移動処理
5000	誰かと出あうか
6000	小人が妖精とあう
8000	敵とあう
10000	戦闘処理

るんだ。だから毎日のゲーム結果をテープやディスケットに保存して，次の日は前日のデータを読み込んでから始める。

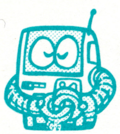
残念ながら，ファミコンでは「メモリー」という厚い壁があって，とてもそこまでのことはできない。ロールプレイングのいろんな要素のうち，基本的なものだけを使って作るミニ・ゲームを作ってみよう。「Tiny ・ ROLL ・ PLAYING」というわけだ。

ホップ・ステップでは，プログラムを少しづつ作ってあとからつけたして行く，というやり方だった。このやり方は，短いファミコンなら大丈夫だけど，将来パソコンで大きいプログラムを作る時には，バグの原因になってしまう。途中で何が何だかわからなくなるんだ。

今回は，まずはじめに，ゲームの中身をきめて，全体の流れを構成する，というやり方でプログラムを作っていこう。



テキスト・ゲーム



まず，メモリー及びグラフィック上の制約（登録してあるキャラクターしか使えない）で，**テキスト・ロールプレイング**（グラフィックを使わないで，文字だけで進行する）としよう。

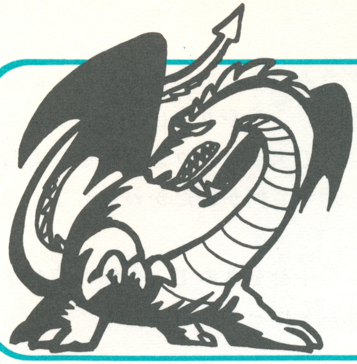
主人公はある地下殿へ侵入，どこかに隠されている王冠を見つけることにしよう。主人公の各要素は，体力，戦闘力，経験値の３つにしよう。持ち物は食料，武器，薬の三つだ。

地下宮殿で出逢うのは，妖精，小人，ゴーレム，兵士，ドラゴンの５種類，妖精と小人は味方で，主人公に何かをくれたり，体力をふやしてくれる。

敵の兵士，ゴーレム，ドラゴンは地下深くなるほど，強さが変化してくる。レベル１から９であらわし基本体力と戦闘力が変化する。

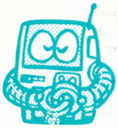
宮殿は地下AFからIFまでの９層。各階は９部屋にわかれる。「A—３」というと地下1Fの３番目の部屋というわけだ。

階段は各階の１と９の部屋にある。



010-2 プログラムの組み立ては

さて，プログラム中で使う主な変数と初期値を図1に書いてみよう。

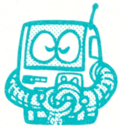


コマンド

プログラムの流れは図2だ。つまりコマンド受付をメインに，コマンドによっていろんな処理をして，またコマンドの受付にもどるわけだ。

コマンドは，荷物を見る「I」，食事をとる「E」，薬を飲む「D」，移動は四方向のカーソルキーだ。

戦闘中のコマンドは，「F」戦うと，「R」逃げるの2種類にしよう。相手をやっつけるか，逃げるか，やられるかのどれかで，始めて戦闘モードから抜け出せる。



初期値の設定

さて**初期値の設定**だ。

●リスト1

```
10 CLS:DIMA(7),B(2,1)
20 FORI=1TO4:A(I)=15:NEXT:A(5)=1:A(6)=1:A(7)=11
30 FORI=0TO2:FORJ=0TO1:B(I,J)=10+I*5:NEXT:NEXT
40 O=RND(20)+80:IFO MOD10=0THEN40
50 B$="ヘイシ ゴーレムドラゴンショウセイコビト "
60 A$="ショクリョウブキレベル クスリ     "
```

図3の各行の働きにしたがってプログラムを組み立てていこう。

100行ではコマンド処理にいく前に主人公の経験値，体力，戦闘力を表示して，プレイヤーがどのコマンドを使うかを決める参考にするところだ。

●リスト 2

```
100 PAUSE50:PRINT"ｹﾞﾝｻﾞｲ ";CHR$(64+A(7)/10);"-";A(
7)MOD10:PRINT"ｹｲｹﾝﾁ=";A(0):PRINT"ﾀｲﾘｮｸ=";A(1):PRIN
T"ｾﾝﾄｳﾘｮｸ=";A(2)*A(5)
```

コマンド入力

さて，いよいよコマンドだ。

200 A=ASC(INKEY$):IFA=0

●リスト 3

```
200 PRINT"ｺﾏﾝﾄﾞﾊ?"
210 A=ASC(INKEY$):IFA=0THEN210
220 IFA<32ANDA>27THEN4000
230 IFA=73THEN1000
240 IFA=69THEN2000
250 IFA=68THEN3000
260 GOTO210
```

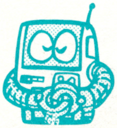
持ち物

1000行からの「I」の処理は，**持ち物**をみるわけだから主人公の持ち物を表示すればいい。

●リスト 4

```
1000 FORI=0TO2:PRINTMID$(A$,I*6+1,6);"=";A(I+4):NE
XT:GOTO100
```

食料と薬



2000行と3000行の「E」と「D」はそれぞれ**食料と薬**をとるわけだが，もちろん薬や食料がなければできるわけがない。もし持っていた場合は，薬で体力を＋4，食料で体力を＋2にしよう。当然それぞれの所持数は－1になるわけだ。

これらの処理をすますと，すぐコマンド処理へもどってもいいが，ここでは，休んでいる間に誰かが来る。と考えて，5000行へ飛ばしてみよう。

●リスト5

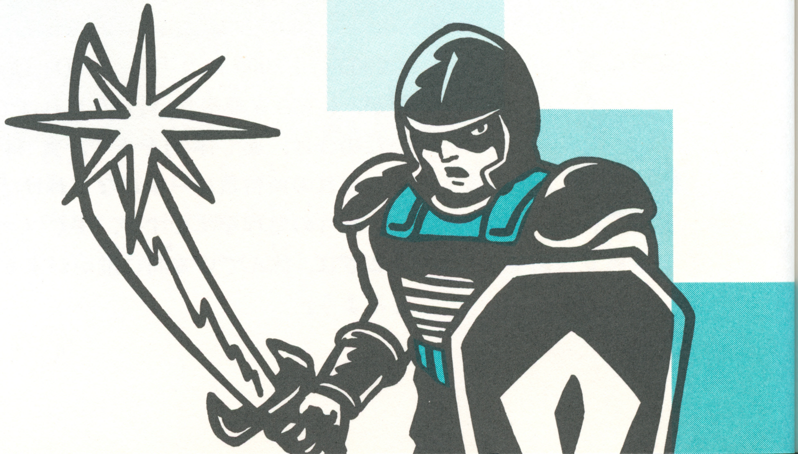
```
2000 IFA(4)<1THEN2500
2010 A(1)=A(1)+2:A(4)=A(4)-1:GOTO3500
2500 BEEP:PRINT"ﾃﾞｷﾅｲ!":GOTO100
3000 IFA(6)<1THEN2500
3010 A(1)=A(1)+4:A(6)=A(6)-1
3500 IFA(1)>A(3)THENA(1)=A(3)
3510 GOTO5000
```

メモリーを節約するため，「デキナイ」という表示は2500行で，また復活した体力が上限値を越えた場合の処理は3500行で一緒にやってしまう。この2500行は次の**移動**処理でも使う。

移動

●リスト6

```
4000 B=A(7)MOD10:IF(B=1ANDA=29)OR(B=9ANDA=28)OR(B<>9ANDB<>1ANDA>29)OR(A(7)>90ANDA=31)OR(A(7)<20ANDA=30)THEN2500
4010 B=0:IFA=28THENB=1
4020 IFA=29THENB=-1
4030 IFA=30THENB=-10
4040 IFA=31THENB=10
4050 A(7)=A(7)+B
```



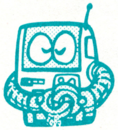
まず4000行で行けない方向を全部2500行へ飛ばしたあと，それぞれの方向へ，主人公の位置A（7）を変更している。

さて，いよいよ5000行だ。

●リスト7

```
5000 IFA(7)=0 THENPRINT"ｵｳｶﾝｦﾊｯｹﾝ!ｵﾒﾃﾞﾄｳ":END
5100 A=RND(10):IFA>4THEN100
5200 PRINTMID$(B$,A*4+1,4);"ｶﾞｷﾀ!"
5300 IFA<3THEN8000
```

王冠の発見



まず5000行で，主人公の位置が王冠の位置と同じなら，**王冠を発見**，めでたしめでたし。これだけじゃつまんない人は，音楽を入れたり，あるいは王冠をもって1Fまで帰ってくる，というようにしてみよう。(Ver, 3じゃないとムリだけどね！)

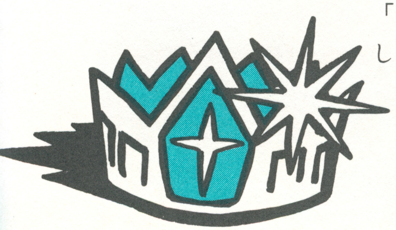
王冠がない場合は乱数で出てくる奴を決めて，まず，妖精が小人なら6000行へ，敵なら8000行へいく。

●リスト8

```
6000 IFRND(3)<>0THEN7000
6100 IFRND(2)=1THENPRINT"ｼｮｸﾘｮｳ":A(4)=A(4)+RND(3)+
1:GOTO100
6200 IFRND(2)=1THENPRINT"ｸｽﾘ":A(6)=A(6)+1:GOTO100
6300 IFRND(3)=1THENPRINT"ﾌﾞｷ":A(5)=A(5)+1:GOTO100
7000 PRINT"ｷｴﾃｼﾏｯﾀ":GOTO100
```

物をもらう

6000行から7000行で小人や妖精が，**薬や食 料をくれたり**，武器のレベルアップをしてくれる。メモリーの関係で，「ショクリョウ」「クスリ」「ブキ」という，ぶあいそうな表示しか出ないが許してほしい。勿論，何もくれないで消える場合もある。





0 1 0 - 3
いよいよ戦闘だ

いよいよ，8000行からは戦闘シーンだ。まず，敵の強さを決めよう。

Cが敵の体力，Dが敵の戦闘力。Eは主人公の体力，Fは主人公の戦闘力だ。Gは，あとで主人公のレベルアップのための数値だ。

●リスト9

```
8000 D=A(7)/10:C=B(A,0)*D+RND(9)-4:D=B(A,1)*D+RND(
9)-4:G=D/10:E=A(1):F=A(2)*A(5)
```

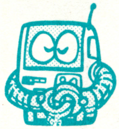
脱出

8050行からコマンドの処理，もし主人公の経験値＋2が敵の体力をうわまわれば，**脱出**成功，ただし，経験値は下がるわけだ。(経験値はマイナスになってもいいことにしよう。)

●リスト10

```
8050 PAUSE50:PRINT "F OR R ?"
8100 B=ASC(INKEY$):IFB=0THEN8100
8110 IFB=70THEN9000
8120 IFB=82THEN8500
8130 GOTO8050
8500 IFA(0)+RND(10)>C THENPRINT"ﾀﾞｯｼｭﾂ":A(0)=A(0)-
RND(3):GOTO100
8600 GOTO8050
```

戦闘



9000行からが戦かいだ。

敵の戦闘力に乱数を足したものと，味方の戦闘力と経験値に乱数を足したものでどっちが相手に打撃をくわえるかを決める。どれだけの打撃をあたえたかは，さっきのレベルGを利用しよう。

敵が倒れたら9500行へ，やられたら10000行のENDルーチンへ飛び，両方の体力が残っていたらまた8050行へもどるわけだ。

●リスト11

```
9000 IFD+RND(50)>F+RND(50)+A(0)THEN9100
9010 PRINT"HIT!":C=C-G-RND(6):GOTO9200
9100 PRINT"ﾔﾗﾚﾀ!":E=E-G-RND(6)
9200 IFE<1THEN10000
9210 IFC<1THEN9500
9220 GOTO8050
```

勝利



敵をやっつけると経験値がGだけふえる。体力の上限値と戦闘力もGに応じて乱数でふえる。現在の体力は，今の戦闘力で残った分だけになる。

そして10000行ではおしまいになる。という処理をすればいいんだ。

●リスト12

```
9500 BEEP:PRINT"ﾃｷｦﾀｵｼﾀ!":A(0)=A(0)+G:A(3)=A(3)+RN
D(G):A(2)=A(2)+RND(G):A(1)=E:GOTO100
10000 BEEP:PRINT"ｱﾅﾀﾊ ｼﾝﾀﾞ!"
10010 PRINT"REPLAY ? Y OR N"
10020 A$=INKEY$:IFA$=""THEN10020
10030 IFA$="Y"THENRUN
10040 IFA$="N"THENEND
10050 GOTO10020
```

途中コマンド入力時に「PAUSE」を指定しているのはスピードが速すぎてキー入力がだぶってしまうためなんだ。

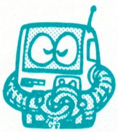


0 1 0 - 4

完成リストの改造法

ロールプレイング・ゲームってこんなものか，と思われるので，一言書かないといけない。

ロールプレイング

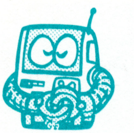


市販されている**ロールプレイング**は，まず主人公の性格や能力を決めたあと冒険に旅立つ，主人公が敵はきれいなグラフィックで表示され，地下の宮殿は迷路や罠に満ちている。

また，お金や魔法を取り入れて，ゲームはより複雑になっている。お金で武器や食料を手に入れたり，仲間を探したり，兵隊をやとったり，ほとんどのゲームがプレイするには何週間もかかる。

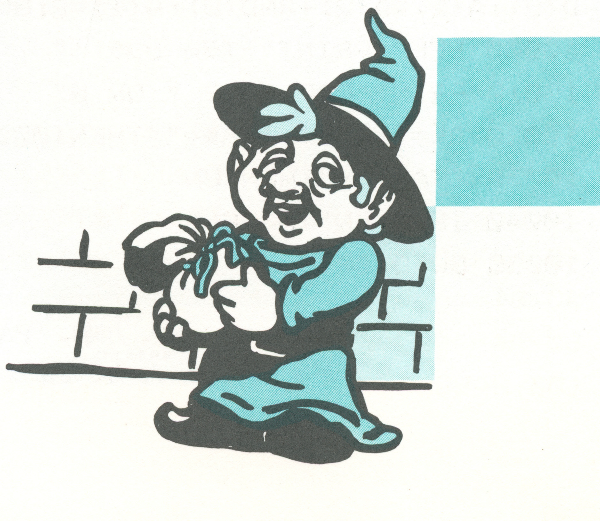
今回のゲームは，ロールプレイングゲームのエッセンスだけを取り上げた「Tiny」版だ。しかし，わずか1982バイトでもここまで出来るんだ。

お金の要素



Ver. 3を持っている人は，**お金の要素**を取り入れたらどうだろう。

敵を倒すとお金が手に入る。妖精は前の通り，何かをくれる味方だが，小人は中立の商人にしてしまう。小人が現われると，小人と



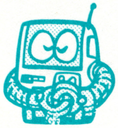
交渉して武器を買うわけだ。

よろい，の要素も入れるとおもしろい。よろいのレベルが上がると，やられた時のダメージが減るというシステムだ。

このように，ロールプレイング・ゲームのシステムを考えていると，次から次へとふくらませていくこともできる。プログラムの勉強にはぴったりかもしれない。

さて，このゲーム，経験値が上がったり，武器のレベルが上がるまではなかなか勝てないかもしれない。**改造法**を教えましょう。

改造するには



まず20行の「A（I）＝15」の「15」を大きくすると，主人公は強くなる。「A（5）」を大きくすると武器のレベルが上がってくれる。

妖精や小人が武器をくれるのが少なすぎると思ったら，たとえば6000行と6300行を入れ変えてもいいし，「RND」の所を変えてもいいね。

妖精・小人・敵が出てくるのは乱数で決めている。5100行の所で「A」が0から4までの値をとると，何かが出てくるわけだ。ここを改造すると，妖精ばかりが出る，なんていうのも可能だ。

もっとも，絶対に勝てるゲームなんてつまんない。うまくいけば勝てる，というのが一番楽しいね。

今回のゲーム，メモリーを気にしながら作ったので，せっかくの音楽機能「PLAY」もなく，表示もすごい簡単になってるけど，まだ残りが数百バイトあるし，みんながそれぞれ改造して，より楽しいゲームにしてくれるとうれしいね。

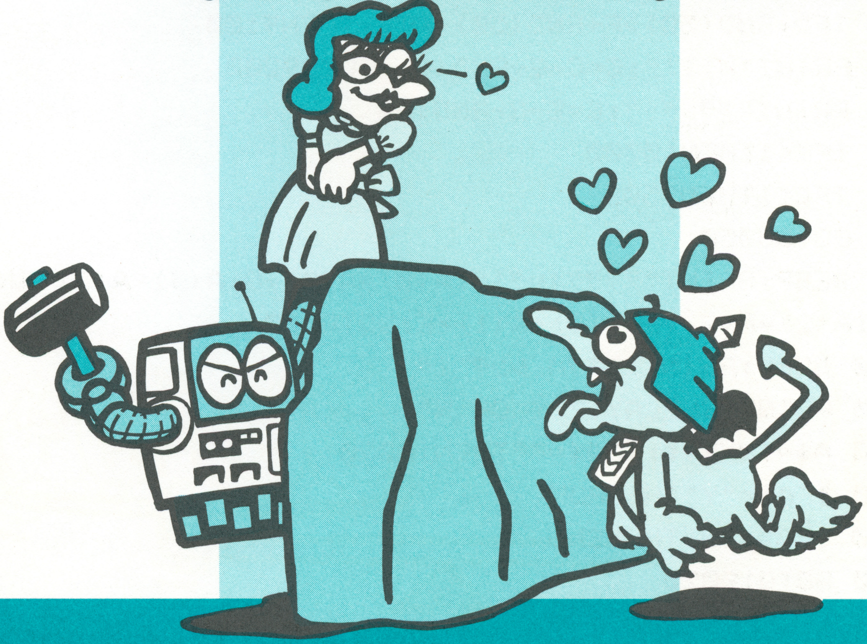
全体のプログラムをもう一度のせよう。(リスト13)いろいろやってみてみよう！

```
10 CLS:DIMA(7),B(2,1)
20 FORI=1TO4:A(I)=15:NEXT:A(5)=1:A(6)=1:A(7)=11
30 FORI=0TO2:FORJ=0TO1:B(I,J)=10+I*5:NEXT:NEXT
40 O=RND(20)+80:IFO MOD10=0THEN40
50 B$="ﾍｲｼ ｺﾞｰﾚﾑﾄﾞﾗｺﾞﾝﾖｳｾｲｺﾋﾞﾄ "
60 A$="ｼｮｸﾘｮｳﾌﾞｷﾚﾍﾞﾙ ｸｽﾘ    "
100 PAUSE50:PRINT"ｹﾞﾝｻﾞｲ ";CHR$(64+A(7)/10);"-";A(
```

```
7)MOD10:PRINT"ｹｲｹﾝﾁ=";A(0):PRINT"ﾀｲﾘｮｸ=";A(1):PRIN
T"ｾﾝﾄｳﾘｮｸ=";A(2)*A(5)
200 PRINT"ｺﾏﾝﾄﾞﾊ?"
210 A=ASC(INKEY$):IFA=0THEN210
220 IFA<32ANDA>27THEN4000
230 IFA=73THEN1000
240 IFA=69THEN2000
250 IFA=68THEN3000
260 GOTO210
1000 FORI=0TO2:PRINTMID$(A$,I*6+1,6);"=";A(I+4):NE
XT:GOTO100
2000 IFA(4)<1THEN2500
2010 A(1)=A(1)+2:A(4)=A(4)-1:GOTO3500
2500 BEEP:PRINT"ﾃﾞｷﾅｲ!":GOTO100
3000 IFA(6)<1THEN2500
3010 A(1)=A(1)+4:A(6)=A(6)-1
3500 IFA(1)>A(3)THENA(1)=A(3)
3510 GOTO5000
4000 B=A(7)MOD10:IF(B=1ANDA=29)OR(B=9ANDA=28)OR(B<
>9ANDB<>1ANDA>29)OR(A(7)>90ANDA=31)OR(A(7)<20ANDA=
30)THEN2500
4010 B=0:IFA=28THENB=1
4020 IFA=29THENB=-1
4030 IFA=30THENB=-10
4040 IFA=31THENB=10
4050 A(7)=A(7)+B
5000 IFA(7)=0 THENPRINT"ｵｳｶﾝﾗﾊｯｹﾝ!ｵﾒﾃﾞﾄｳ":END
5100 A=RND(10):IFA>4THEN100
5200 PRINTMID$(B$,A*4+1,4);"ｶﾞｷﾀ!"
5300 IFA<3THEN8000
```

```
6000 IFRND(3)<>0THEN7000
6100 IFRND(2)=1THENPRINT"ｼｮｸﾘｮｳ":A(4)=A(4)+RND(3)+
1:GOTO100
6200 IFRND(2)=1THENPRINT"ｸｽﾘ":A(6)=A(6)+1:GOTO100
6300 IFRND(3)=1THENPRINT"ﾌﾞｷ":A(5)=A(5)+1:GOTO100
7000 PRINT"ｷｴﾃｼﾏｯﾀ":GOTO100
8000 D=A(7)/10:C=B(A,0)*D+RND(9)-4:D=B(A,1)*D+RND(
9)-4:G=D/10:E=A(1):F=A(2)*A(5)
8050 PAUSE50:PRINT "F OR R ?"
8100 B=ASC(INKEY$):IFB=0THEN8100
8110 IFB=70THEN9000
8120 IFB=82THEN8500
8130 GOTO8050
8500 IFA(0)+RND(10)>C THENPRINT"ﾀﾞｯｼｭﾂ":A(0)=A(0)-
RND(3):GOTO100
8600 GOTO8050
9000 IFD+RND(50)>F+RND(50)+A(0)THEN9100
9010 PRINT"HIT!":C=C-G-RND(6):GOTO9200
9100 PRINT"ﾔﾗﾚﾀ!":E=E-G-RND(6)
9200 IFE<1THEN10000
9210 IFC<1THEN9500
9220 GOTO8050
9500 BEEP:PRINT"ﾃｷｦﾀｵｼﾀ!":A(0)=A(0)+G:A(3)=A(3)+RN
D(G):A(2)=A(2)+RND(G):A(1)=E:GOTO100
10000 BEEP:PRINT"ｱﾅﾀﾊ ｼﾝﾀﾞ!"
10010 PRINT"REPLAY ? Y OR N"
10020 A$=INKEY$:IFA$=""THEN10020
10030 IFA$="Y"THENRUN
10040 IFA$="N"THENEND
10050 GOTO10020
```

作戦指令
011
ランダムテクニック



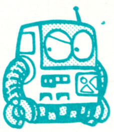
0111-1



ものぐさな君への打ち込み方法

もうこれで，君はファミコン・ベーシックのすべてを自分のものにしたわけだけど，キーボードのキーの入力は速くなっただろうか。何回もいろんなプログラムを入力していると自然にスピード・アップしてくるものだけど，いまだに，ウロウロとキーボードを探しているかもしれないね。「RESTORE」なんて長い単語になるとけっこう入力するのに時間もかかるしね。

短縮入力



今までふれなかったけど，マニュアルなんかでもう，知っているかもしれないが，これらの命令のほとんどは，「短縮形」といって，もっと少ないキー入力で打ち込めたんだ。ごめん。たとえば「RESTORE」は「RES.」のようにね。

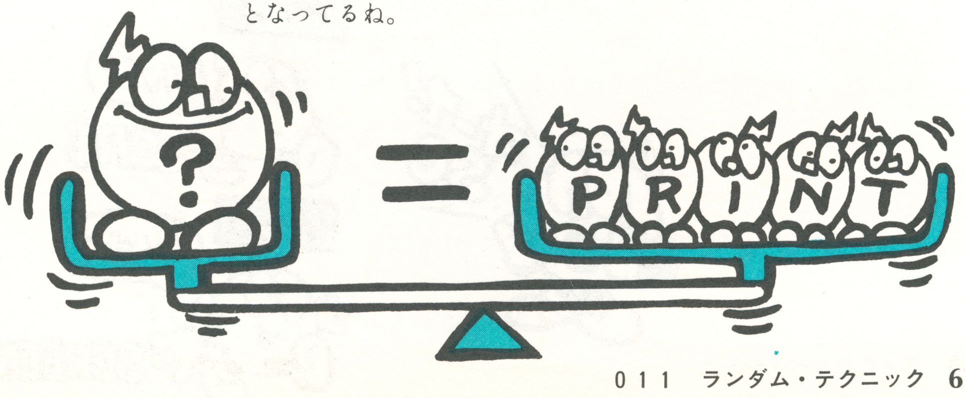
なんで今までその説明をしなかったかというと，この短縮形，コンピューターによってその縮め方が大部違うし，短縮形を使用することで，本来の命令の意味がぼやけてしまうと思ったからだ。

```
10 ? A:G. 20
```

と入力してLISTをとると

```
10 PRINT A:GOTO 20
```

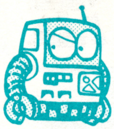
となってるね。



「PRINT」が「？」，「REM」が「 ' 」で代用する以外は，これらの短縮形はその命令の頭文字のいくつかを書いたあと「.」をつけることで代用できる。

それぞれの命令がどういう短縮形をもつかは巻末の総索引で，読み方と一緒に書いてあるので参考にしてほしい。

問題点

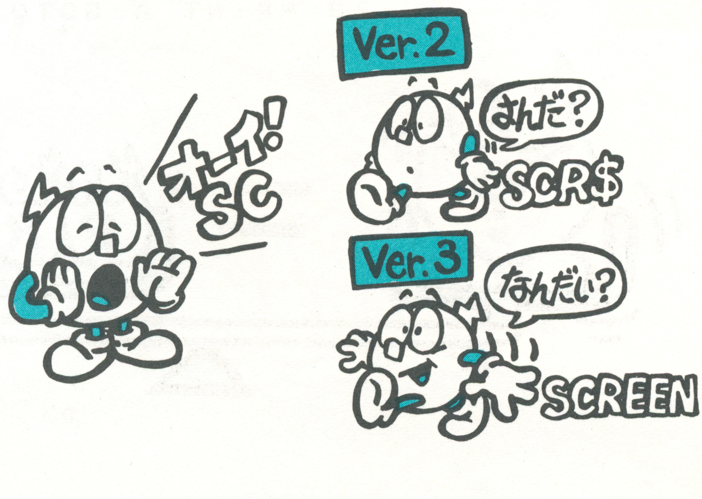


ただ注意点がいくつかある。

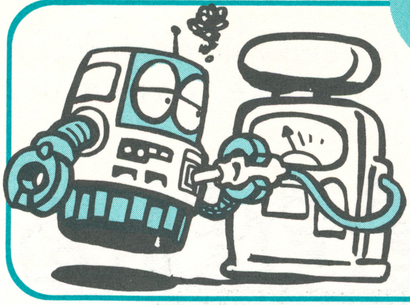
「POSION」の短縮形は「POS.」だが，知っての通り，「POS」という別の関数がある。同じく，Ver. 3の「ERROR」の短縮形は「ERR.」だが，「ERR」とすると別の関数になってしまう。他に「STI.＝STICK」だけど「STR.」は，「STR$」で「STRIG」は「STRI.」だったり，けっこう間違いやすいものが多い。何がなんでも短縮形にしようとして，「CGEN」⇒「CGE.」なんて，結局ストローク数（キーを打つ回数）が同じのがいっぱいあるし，かえって間違いそうだ。

一番ひどいのは，「SC.」はVer. 2では「SCR$」のことなのにVer. 3では「SCREEN」の短縮形になっている。Ver. 2とVer. 3のくい違いはバグ情報（ステップ編）でもふれたけど，ほんとに同じ人達が作ったんだろうかね？

と言っても，便利なものもいっぱいあって「L.」や「S.」や「V.」，「？」，「 ' 」，「G.」なんかはよく使う。それぞれが何にあたるかは自分で調べようね。なお，他のものは「？」→「PRINT」のように置き変えられるけど「 ' 」だけは，もとのままの表示だ。「 ' 」は「REM」の別の形だと思ってほしいんだ。



011-2 メモリーを節約するには



ファミコンでプログラムを作ることはメモリーとの戦いだっていうのはもう君達も実感しているかもしれないね。

君達の大先輩達も昔は同じ戦いをしていたんだ。まだパソコンと呼ばれるものがなかったころ，パソコンの原形はベーシックすら走らない少メモリーだったんだ。でも，その中で先輩達はいろんな工夫をこらして，おもしろいゲームをいっぱい作ったんだ。君達もがんばらなくちゃ。

```
10 IFA=0THENPRINT"A"
10 IF A=0 THEN PRINT "A"
```

NEW したあと上の二つを入力のあと，それぞれダイレクトで，「PRINT　FRE」としてみよう。1967と1963で上の方が残りバイトが多いね。

スペースつめ



二つの差の4バイトはどこにあるだろう。気がついたかな。スペース「空白」が下の方に五つあるね。そのうち，行番号の直後の一つ以外を上の方はつめているんだ。つまり4個の空白をけずったことによる差なんだ。

この行番号の直後の空白は，もし，つめて入力してもリスト表示の時は自動的に挿入されるし，メモリーには関係がない。

空白つめが，ファミコンのメモリー節約の第一歩だ。ただし，注意しなければならないのは，

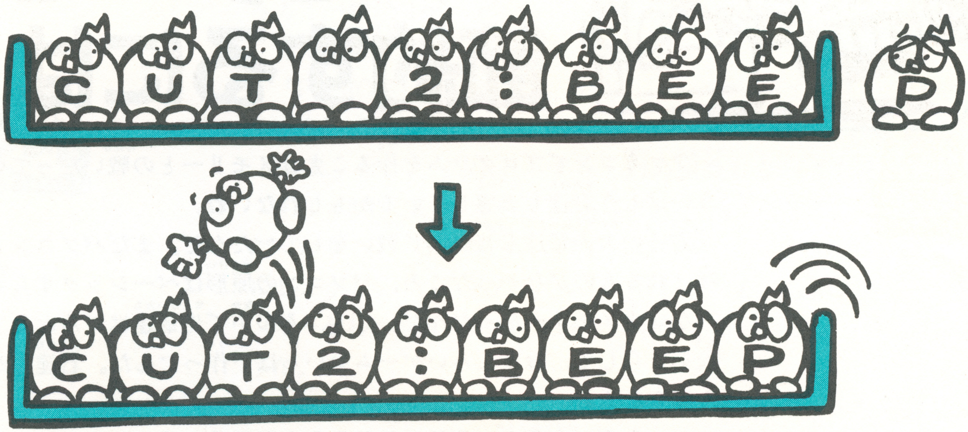
```
IFA=BTHEN
FORI=A1TO
```

のように，変数名（BやA1）と次の単語の間の空白をとってしまうと，

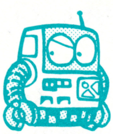
```
IF A=BTHEN
```

FOR I=A1TO

のように，「BTHEN」や「A1TO」という名前の変数だと間違われてしまうので，こういう場合は必ずスペースを入れなければいけないんだ。



マルチ・ステートメント



次に有効なのは，マルチステートメントだ。「：」で命令をつなげることができるのを利用する。NEW したあと，

```
1Ø  A=1          ⎫
2Ø  PRINT  A     ⎭ ①
1Ø  A=1：PRINT  A    ②
```

のそれぞれの残りバイトをとると，1966と1969になる。行を新しく一つ作るのには4バイトを消費する。

一つの行にまとめることで－4，変わりに「：」を使うから＋1で3バイトの節約になる。「GOTO」や「GOSUB」で飛んでこない所は，できるだけ一つの行へたくさんの命令をつめ込んだ方がいいね。ただし，

```
1Ø  GOTO  3Ø
2Ø  A=1
3Ø  PRINT  A
```

を，

```
2Ø  A=1：PRINT  A
```

として30行をけずると，10行の「GOTO」の行き先がなくなりエラ

ーになる。一度作ったプログラムを書き直す時は，よく注意しようね。

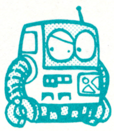
もう一つ，ファミコンの特色で

```
IF A=Ø THEN PRINT A
```

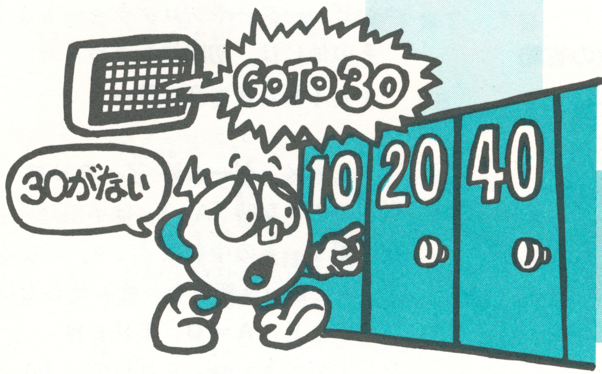
を

```
IF A=Ø PRINT A
```

THENの省略



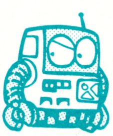
のように「**THEN**」を省略しても「THEN」以下の文章を実行してくれる。これは他のパソコンには通用しない（一部のパソコンでは通用するがメモリーは同じ）方法なんだ。



以上にあげたような手法は，メモリー消費をおさえるには有効だが，プログラムは非常に見づらく，もしバグがあったとしたら，それを探すのにも大変になってしまう，という反面がある。

それに対し，プログラム手法によってメモリー節約をはかる方法がある。

サブルーチン



同じような処理をする場合はできるだけ**サブルーチン**を使い，またFOR・NEXTループで効率化をはかる。ステップ編でのべたように，サブルーチンへ受けわたす変数の中身を変えることで，似たような処理を一つのサブルーチンで処理する方法だ。

ただし，サブルーチン化で，かえってメモリーを使う時もある。

```
10 CLS
 ～
50 CLS
```

を

```
10  GOSUB 90
  ～
50  GOSUB 90
90  CLS:RETURN
```

などとするのはバカげているのはわかるだろう？

このサブルーチン化は，デバッグの時にプラスになることが多い。似たような処理を一ヶ所でやっていると，その処理に間違いがある時，その場所を探しやすい。また，一回あるゲームで作ったサブルーチンを他のゲームで利用するのも可能だ。このやり方は，大型のコンピューターやプログラマーもよく使う。

初期値の省略



その他には，**初期値**をきめる時

```
10  A=Ø:B=1
        ～
××  ゲーム終了:RUN
```

などのような時「A=Ø」は不用だ。新しく変数を作る時，特に定めない限り，その中身は「　Ø」になる。

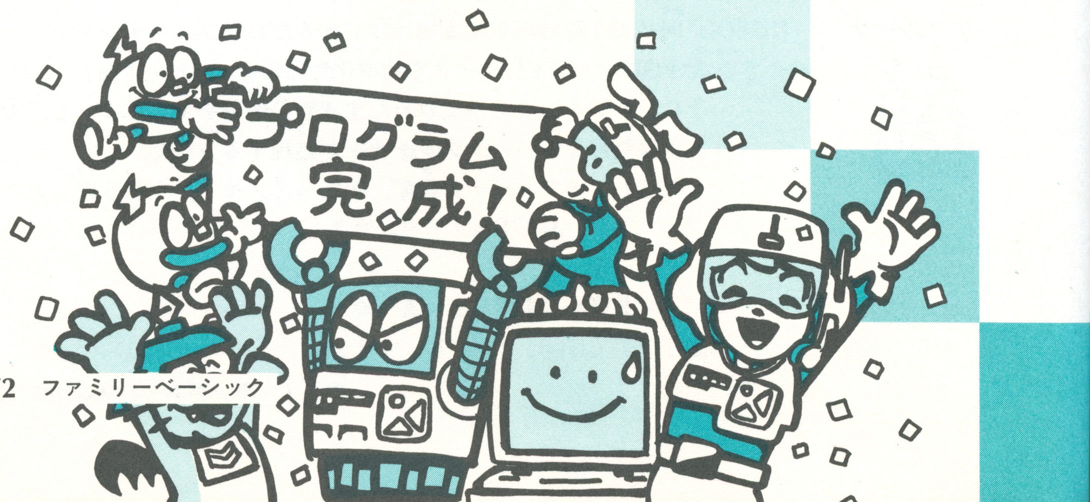
「A」という変数を一度も使わないまま突然

```
IF A=Ø THEN
```

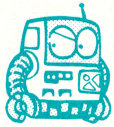
などとでてきた時，それ以前に「A=Ø」と代入してなくとも「A」という変数が出てきた時，必ず「A」の中身は「 Ø」から始まるわけだ。

また，次のプログラムを実行しよう

```
10  B=1:A=(B=Ø):PRINT A
20  B=Ø:A=(B=Ø):PRINT A
```

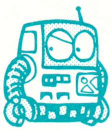


真と偽



ファミコンの内部では，あることがらが**真実**の時は「－１」，**間違っている**時は「Ø」で処理している。「A＝（B＝Ø）」というのは「B＝Ø」というのが間違っているか正しいか（つまりØか－１）の値を「A」に入れろという命令だ。したがって上のプログラムの実行結果は「Ø」，「－１」になる。

IFの省略



ファミコンの場合の**IF文**は，「IF～THEN」の「～」の部分が間違っている（Ø）の時は「THEN」以下を実行しない。それ以外（Øでない数値）の時は実行する，という仕組みになっている。そこで，

```
IF A<>Ø THEN PRINT "B"
```

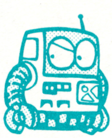
は

```
IF A THEN PRINT "B"
```

と書いても同じ働きをする。

パソコンクラスになると長いプログラムが多いので，変数の名前もある程度，意味のわかるような長い変数名を使うことが多いが，ファミコンでは使う変数もそれほど多くないし，短い変数名，それも一文字で十分だろう。変数名が一文字長いごとに，１バイト，それもプログラム内に一回でてくるたびにメモリーは減って行くからだ。

文字変数



また，**文字変数**はできるだけ使わない方がいい。

```
1Ø A$="TEST":B$="PROGRAM"       ┐
2Ø PRINT A$:PRINT B$            ├ ①
3Ø PRINT FRE                    ┘
1Ø A$="TESTRROGRAM"             ┐
2Ø PRINT LEFT$(A$,4):P          │
   RINT RIGHT$(A$,7)            ├ ②
3Ø PRINT FRE                    ┘
```

の二つをくらべてほしい。残りバイトは28バイトも②の方が多いだろう。

ファミコンでは文字変数を１個使うごとに，その文字変数の中身の長さには無関係に，だまって31バイトをその文字変数の中身を記録する場所としてとってしまうのだ。

パソコンは，文字変数１個が使用する場所の大きさは，その時，

その変数の中身の文字列の長さできる（可変長）なのだが，ファミコンは処理を簡単にするため，長さを変えない（固定長）なのだ。
　だから，文字変数はできるだけ使わずに，使っても，その使用回数が少なければ上記のように一つの変数を共有した方がいい。もし，「PRINT」命令だけなら，

```
1Ø PRINT "TEST"
2Ø PRINT "PROGRAM"
3Ø PRINT FRE
```

とするのが一番いいだろう。
　だから

INKEY$

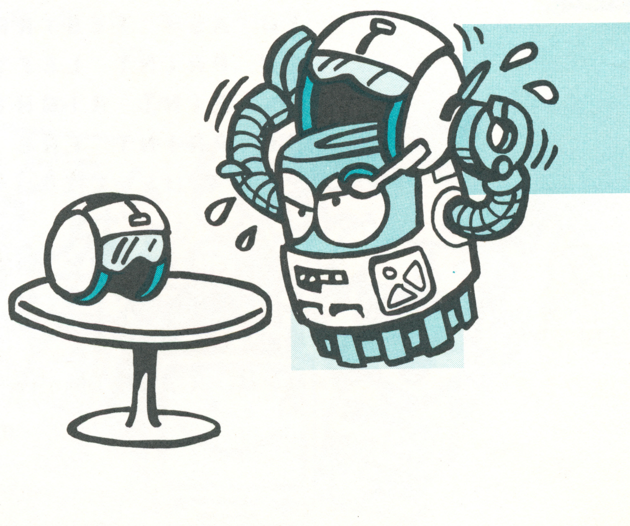


```
1Ø A$=INKEY$:IF A$="" T
   HEN 1Ø
```

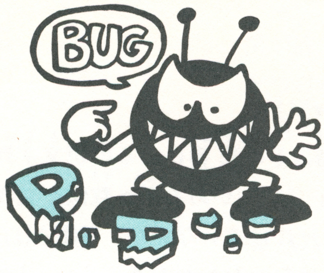
なども

```
1Ø A=ASC(INKEY$):IF A=
   Ø THEN 1Ø
```

とした方が，ずっと節約できる。(プログラム自体は下の方が多きいが，RUN したあとで FRE をとると，上の方がたくさんメモリーを使っているのがよくわかる)
　数値変数も，影響を及ばはない時は同じ変数名を使った方がいいよ。
　これらのテクニックを使えば，普通の方法で書くと，2500バイト近くかかるプログラムも，1982バイトまで縮めることも可能だ。

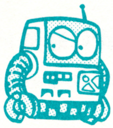


011-3



Debug法のいろいろ

プログラムを作っていると，必ずぶつかるのが，「バグ」だ。日本語になおすと，「虫」。プログラムの中にひそんで，プログラマーの予定していない動きをしてしまうんだ。この虫を探して，やっつける。つまり正しいものになおして行く作業を「デバッグ＝Debug」という。このバグには二通りある。「エラー・メッセージ」がでてプログラムストップする場合と，プログラム自体は働いているんだが，思ったような動作をしない場合だ。

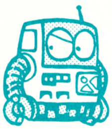


エラーメッセージ

エラーメッセージがでるような場合は，デバッグはわりあい簡単だ。巻末のエラーメッセージ表を見ながら考えよう。

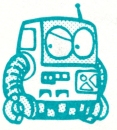
かけた一組

まず，「NF」と「RG」，Ver. 3では「NR」，「RE」エラー。これらはそれぞれ，「FOR～NEXT」や「GOSUB～RETURN」，「ON ERROR　GOTO～RESUME」のように**一組**になって使う命令の片方がないためにおこるエラーだ。だから欠けている命令をたしてやればいいんだ。ただし，サブルーチンをメインルーチンのうしろにおいた時，



```
1Ø  GOSUB  8Ø
2Ø  PRINT  A
3Ø  END
8Ø  A=1:RETURN
```

このようなプログラムで30行がないと「RG」エラーになる。命令としては「GOSUB～RETURN」と1SETになっていても，メイン（10，20行）を実行したあと，30行がないとそのまま80行へ行ってしまうためだ。



文法のミス

「SN」エラーは，最も多いかもしれない。命令や関数のスペル（つづり）を間違ったり，**文法が間違ったり**した時だ。その行を表示し

て，間違いをなおせばいいんだ。

これも

```
10 READ A
20 DATE "ABC"
```

のように，数値変数を読みに行く時，その順番にあるのが文字列だと「SN」エラーとなる。たしかに，文字列がある所へ数値を読みに行けという命令だから「SN」エラーには違いないが，「READ」文と「DATE」文がはなれている時にはなかなかわかりにくい。「データ・タイプ・ミスマッチ」エラーとか表示して，「READ」文と「DATA」文，両方の行番号がわかるとありがたいものだが。

「READ」文で読みに行く回数より「DATA」の個数の方が少ない時は「OD」エラーだ。

数の制限



スプライトの座標指定など，パラメーターの**数値の指定に制限**がある場合，その制限を越えたものを指定しようとするとおきるのが「IL」エラーだ。

```
LOCATE X,Y
```

という所でこのエラーがおきると，ダイレクトで

```
PRINT X,Y
```

として，どちらの変数が制限を越したか，そして，どこで越したか調べなければいけないんだ。



この，制限を越えるというエラーは他にもある。演算の途中や結果が許容範囲を越えた時におきるのが，「OV」エラー。ただし，気をつけなければいけないのは，2バイトで正（＋）負（－）の整数を表現するため，－32768から32767までしか扱えないにもかかわらず，かけ算やわり算ではチェックをしていないんだ。

```
PRINT 32767+1
```

は「OV」エラーになるのに，

```
32768*1=-32768
32768/128=-256
200*200=-25536
```

のようにおかしな（コンピューターの性格を知ってるとあたりまえなんだが）結果になってしまう。これも，一つのバグといえるね。

「OM」はメモリーが足りなくなったエラー。「 RUN 」する前はメモリーがあまっていても，実行するとこのエラーが出るのは変数が多すぎるせい。前章のテクニックでメモリーを減らすしかないんだ。

エラーが出た時点で「PRINT　FRE」をして残りが多い時はステップ編　31Pのような場合もある。

「GOTO」，「GOSUB」，「RESUM」，「THEN」のあとに行番号を指定しているのにその行番号がないのが「UL」エラーだ。

配列変数で，同じ配列「A（ ）」を2回も「DIM」を使って宣言しようとすると，「DO」エラー。

「DIM」で宣言してなかった配列変数を使ったり，添字（配列変数のカッコの中の数字）が，最初の宣言より大きな数字だったりすると「SO」エラーになる。とマニュアルに書いてあり，パソコンもそうなんだが，なぜか私のもっているカセットはVer. 2もVer. 3も「IL」エラーになってしまう。なぜでしょう？

「DZ」エラーは「Ø」で割り算をした時，「TM」エラーは「＝」の右辺と左辺のタイプ（文字か数値か）が違う時にでる。

「ST」エラーは文字列の文字が32文字を越した時にでるんだが，「INPUT」や「READ」で読み込む時にだけでて，たとえば，長さ20文字の文字列を「A$」とした時

```
A$+A$
```

のようなたし算をすると「OV」エラーになる。

「T」が何重にも複雑になったり，ファミコンの能力を越える式に対しては「FT」エラーになる。ただしこれも，実験の結果，ダイレクトで

```
PRINT  ((((((… (1+1)) …)))))
        └─────┘     └─────┘
          15個         15個
```

のところでファミコンはウンともスンとも言わなくなって，エラーメッセージも返してこなくなった。

「CC」エラーは「STOP」命令や「STOP キー」でプログラムの実行を中断した時に，「CONT」で再開しようとしても，できない時に表示される。

「TP」エラーはテープを読み込もうとしている時にミスがおきた時にでる。

「MO」エラーは普通のパソコンでは

```
LOCATE 3,
A=
```

のようにうしろに書かなければいけないものを書きおとした時にでるエラーだが，ファミコンでは「SN」エラーと表示されてしまう。どうやったら「MO」エラーになるのだろう。

同じく「VF」は未定義の関数を呼ぶ時にでるエラー。パソコンではいろんな関数をプログラマーが自分で作れるからこのエラーの意味もあるが，ファミコンでは，ユーザーが定義できる関数は「DEF SPRITE」と「DEF MOVE」だけだ。そこで「DEF ～」で定義しないまま「MOVE 3」とか「CAN 4」とかやっても，このエラーはでなかった。

どうも普通とは違う使い方をしているのか，マニュアルの説明が間違っているのか，我々には原因がつかまえられなかった。すみません。

「NB」エラーは「BGGET」をしていない時に「BGPUT」をする，つまり BG データがないのに BG 画面へデータを転送しようとした時のエラーだ。

これらの登録されたエラー以外のエラーがおきた時にでるのが，「UP」エラーなんだ。

メッセージのでないミス



どうもこれらのエラーメッセージについては，いまひとつ，パソコンと同じ単語を使いながら，きれいに整理されてないようだが，それでもエラーメッセージがでるとデバッグはまだ容易だ。

もちろんファミリー・ベーシック自体のバグ（ホップ・ステップ・ジャンプ各編にのっている。巻末の索引を見てほしい）の場合はほとんどお手上げに近いんだが，プログラム・ミスで，エラーメッセージがでない時も，けっこうやっかいだ。

たとえば

```
A=A+2
```

とするのを

```
A=A*2
```

としたとしよう。この場所を何回も通りAの値が次第に大きくなっていく。前述のようにかけ算は32767から突然マイナスになる。それでも「OV」エラーになったり，このあと

```
LOCATE A,B
```

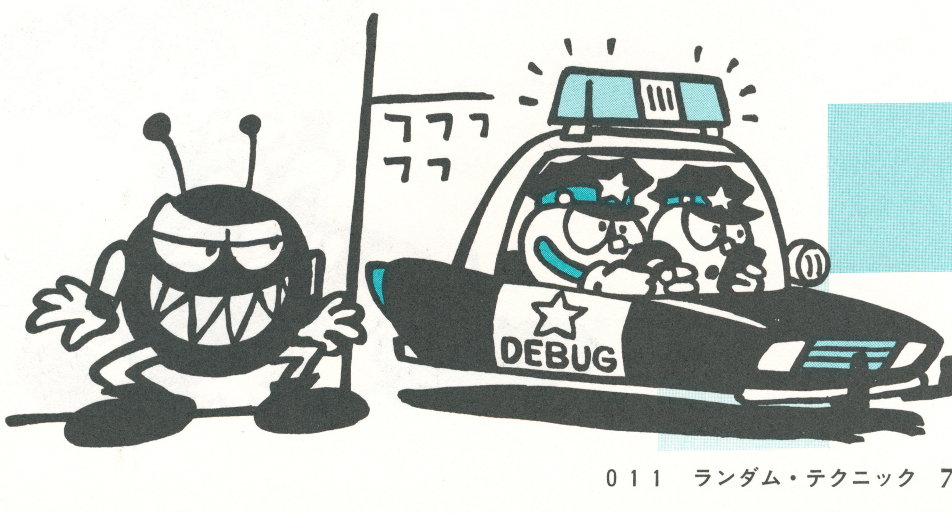
のような場所があれば「IL」エラーでプログラムが止ってくれる。ところが，もとのAが「3」でここを一回しか返らないと

```
LOCATE 5,B
```

のはずが

```
LOCATE 6,B
```

になってエラーにならない。



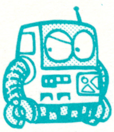
このような時，Ver. 3では

FIND "A"

として，「A」という変数を使っている場所をしらみつぶしに調べて行けばいいが，Ver. 2ではこの「FIND」命令がないため，LISTを見ながら根気よく調べるしかないんだ。

実はDebug法には，「これだ！」という方法はない。前章の「メモリー節約」とは逆に，必ず空白を入れ，見やすいプログラム，そして論理的なプログラム作りをする。それでもバグがでたら，外で運動したり，ピアノを弾いたり，**冷却期間**を置くとわかったりすることも多い。

頭を冷やして



ともかく，繰り返し，繰り返し，プログラムを見なおし，あやしい変数を調べることだ。バグのすみからしき前後に「STOP」命令を置き，止まった所で各変数の値を調べたり，「TRON」でプログラムの流れを追うのも有効な手段になるよ。

バグを退治できるようになる，というのは，実はプログラム能力があるということなんだ。プログラムの能力がなければ，バグがあることすらわからないからね。

プログラムを作っている以上，バグとのつき合いは絶対なくならない。お友達になってしまうのも一つの手だ！





011-4

コンピューターの言葉 2進数と16進数

今まで,「メモリー」を言い表わす時に,何回も「バイト」という言葉を使ってきた。電源を入れた直後に

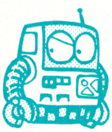
1982 BYTES FREE

と表示される「BYTES」だ。英語では

単数形　1 BYTE

複数形　2 BYTES

BYTE



と使いわけているが,日本語では両方とも「**バイト**」と呼んでいる。

この「バイト」とは何だろう。

コンピューターは電気で働いている。何かをどこかへ記憶している時,それを読み出すのも電気だ。一番簡単な方法は,一つ一つの場所にスイッチをつけて,そのスイッチが「ON」か「OFF」かを設定できるようにすればいいんだ。

図1を見よう。コタツの手元スイッチがいっぱい並んでいる。「Sw1」が「ON」,「Sw2」が「OFF」。コンピューターの内部はこうなっている。このスイッチが何万,何十万と並んでいるわけなんだ。

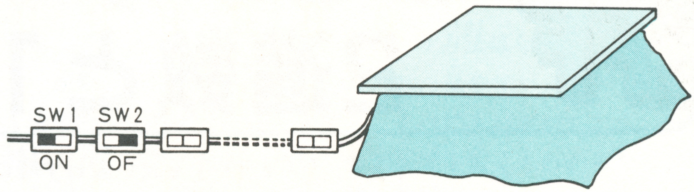
このスイッチが「ON」の時を「1」そして,「OFF」の時を「Ø」と呼んでみる。(「High」「Low」という分け方もある)。とするとコンピューターの内部は

1ØØØ1Ø111Ø〜

という長い(何十万ケタもの)数字で表現できることになるわけだ。

このスイッチ1個分のことを「**BIT**」と呼ぶ。英語の「少し」とか「一口分の」とかいう意味だ。言い変えれば,コンピューターは何

●図 1



十万ビットのスイッチの固まりだ。この何十万ビットの長い数字を，８個ごとにひとまとめにしたのが「BYTE」なんだ。つまり，

1=8?



1 BYTE ＝ 8 BITS

のことだったんだ。７個や10個で区切っちゃいけないのか，と言うと，そんなわけでもないのだが，あとでのべるように，一つの区切りとしては丁度よかったのと，アメリカの大コンピューターメーカーが，「１バイトは８ビットにしますよ」と決めたのに，同調したため，こう決ったんだ。だから，ごく一部の会社の機械では，「１バイト≠８ビット」ということもあるけど，ほとんどのコンピューターは「１バイト＝８ビット」だ。

さて，この１ビットに数字を記憶するためにどうすればいいんだろう。一番簡単なのは，１バイトにある「１」の個数をその数とする方法だ。(図２)。ところが，この方法では，Øから７までしか１バイトに入らない。

次に，この８ビット，つまり８個のスイッチに１から８までの数字をつけて，その数字の所のスイッチが「１」なら，その数字をたしたものを，その数としてみよう(図３)。今度はØから36までの数値が１バイトに入る。でも，これも少し無駄である。たとえば「14」を入れるにもいろんな方法があるのがわかるだろう（図３)。

まったく無駄のない，ということは一通りの数字を表わすのにはたった１つのスイッチの並びしかない，というには，どうすればいいだろうか。そこで考えられたのが，８個のスイッチに１から128までの数値をあてはめたものだ（図４)。このやり方だと，Øから255

●図 2

Ø =	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1 =	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
7 =	Ø	1	1	1	1	1	1	1
8 =	1	1	1	1	1	1	1	1

●図 3

	8	7	6	5	4	3	2	1
Ø =	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7 =	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø
14 =	Ø	1	1	Ø	Ø	Ø	Ø	1
=	Ø	Ø	1	Ø	0	1	Ø	1
36 =	1	1	1	1	1	1	1	1

●図 4

	128	64	32	16	8	4	2	1
Ø =	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
7 =	Ø	Ø	Ø	Ø	Ø	1	1	1
113 =	Ø	1	1	1	Ø	Ø	Ø	1
255 =	1	1	1	1	1	1	1	1

の256通りの数値が表現できるんだ。

この方法が，8ビット＝1バイトで一番たくさんの数を表現できる方法なんだ。というのもあたりまえの話しで，これは，「2進法」という数の数え方の書き方なんだ。

2369

という数字を僕達は何気なく使っているけど，これは「10進法」という数の数え方を使っているんだ。

$1000 = 10 \times 10 \times 10 = 10^3$（10の3乗）

$100 = 10 \times 10 = 10^2$（10の2乗＝自乗）

$10 = 10 \times 10^1$

$1 = 10^0$

「XのY乗」というのを学校で習っただろうか。これは，「X」を「Y回」かけあわす。という意味だ。「6」の「4」乗とは

$6 \times 6 \times 6 \times 6$

のことで

6^4

と書くことは知っているね。そして「Y＝Ø」の時は必ず

$X^0 = 1$

にするという約束なんだ。

この考えを使うと，さっきの「2369」という数値は，

$2 \times 10^3 + 3 \times 10^2 + 6 \times 10^1 + 9 \times 10^0$

と考えられているね。

「8」に「1」をたすと「9」になるのに，「1」をたすと「1Ø」という2ケタの数になるのを不思議に思った経験はないだろうか。不思議なのはあたりまえで，これは，「1Ø」を一つの束として数えるよ，という約束事がある。というだけの話だったんだ。

数の数え方は「10進法」に限るわけではない。「1時間＝60分」は「60進法」，「1日＝24時間」は「24進法」，英語は「TEN＝1Ø」の次は「ELEVEN＝11」で，「ONE」という単語がでるのは「2Ø＝TWENTY」の次の「TWENTYONE＝21」，これなんかも「20進法」のなごりだ。

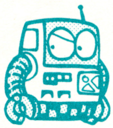
ここで，図4をもう一度見てみよう

$1 = 2^0$

$$2=2^1$$
$$4=2\times2=2^2$$
$$8=2\times2\times2=2^3$$
$$16=2\times2\times2\times2=2^4$$
$$32=2\times2\times2\times2\times2=2^5$$
$$64=2\times2\times2\times2\times2\times2=2^6$$
$$128=2\times2\times2\times2\times2\times2\times2=2^7$$

ということがわかる。

同じ数字も



「1Ø1」という数字がある。

1Ø進法$=1\times10^2+1\times10=101$
16進法$=1\times16^2+1\times16=257$
6Ø進法$=1\times60^2+1\times60=3601$
2進法$=1\times2^2+1\times2^0=5$

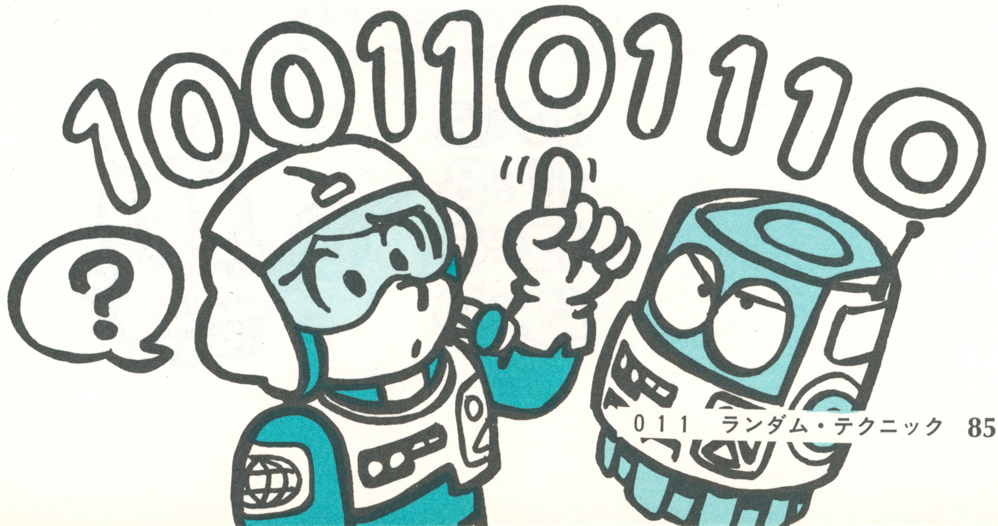
スイッチが「ON」な「OFF」かで内部表現をしているコンピューターにはこの「2進法」がぴったりだ。しかし，一つの大欠点がある。たとえば「10進法」の「256」を「2進法」で表わすと

「1ØØØØØØØØ」

と9ケタにもなる。「832456」なんていう数字を2進法になおすと，何ケタになるだろう。第一，人間が，

1ØØ11Ø111ØØ11ØØ11

なんて数字を見て，どのくらいの大きさか判断できない。かと言って，全部をどうなっているか想像もつかない。そこで考えだされた



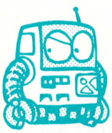
のが「16進法」というものだ。

まず，1バイトの8ビットを4ビットづつの二つの固まりにわけてみよう。そうすると

ØØØØ	＝Ø	
ØØØ1	＝1	
ØØ1Ø	＝2	
ØØ11	＝3	
Ø1ØØ	＝4	
Ø1Ø1	＝5	
Ø11Ø	＝6	
Ø111	＝7	
1ØØØ	＝8	
1ØØ1	＝9	
1Ø1Ø	＝1Ø	＝A
1Ø11	＝11	＝B
11ØØ	＝12	＝C
11Ø1	＝13	＝D
111Ø	＝14	＝E
1111	＝15	＝F

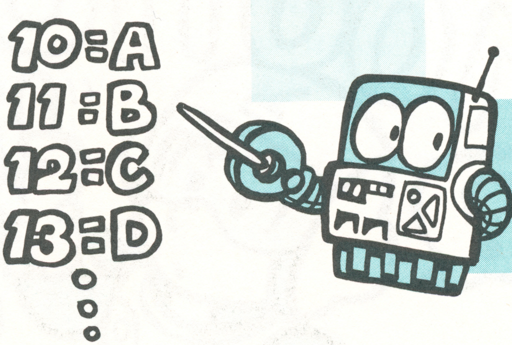
の16通りだということがわかる。ここで，革命的考えを入れてしまうのだ。

10はA?



「1Ø」のことを「A」,「11」のことを「B」…「15」のことを「F」と呼ぼう。

こうすると，4ビットで表わされる数値は「Ø」~「F」までの1ケタの数値で表現できるだろう。そして1バイト，つまり8ビット



の数値は，4ビットづつ1ケタで表わせるから「ØØ」から「FF」まで，つねに2ケタで表現できるわけなんだ。たとえば

```
1 1 Ø 1   1 Ø Ø Ø
   ⇩         ⇩
1 3 =D        8        = 1 3 × 1 6¹+ 8 × 1 6⁰= 2 1 6
   └────────┘
        ↓
      D 8  =  D  × 1 6¹+ 8 × 1 6⁰= 2 1 6
              ⇩
             1 3
```

これを「16進法」と呼ぶわけだ。

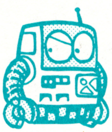
コンピュータで使う時には「10進法」で表わした数字の区別をつけるために，「16進法」で表した数字の前に必ず「&H」というマークをつける（パソコンによっては「$」を使うものもある）

&H3Ø＝3×16¹＝48

3Ø＝3×1Ø¹＝3Ø

というわけだ。そして，「10進法」で表わされた数値を「16進法」になおしてくれるのが，ただ一つ説明してない関数「**HEX$**」だ。

HEX$

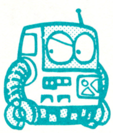


A$＝HEX$（48）：PRINT　A$

で「30」となるはずだ。ただし，この「HEX$」で教えてくれるものは「文字」の形のため，必ず「＝」の左辺は文字変数でなければいけない。

また，一度文字に変わったものを数値に戻すには，

VAL



A＝VAL（〝&H〞＋A$）

のようにしなければならない。もちろん直接なら

A＝&H3Ø

としても OK だ。

この「&H」を読む時は，「&H20」なら，「あんどえいちにじゅう」とか「ヘキサにじゅう」のように読むとよい。2進法は「BIN$」（バイナリー）ともいい。パソコンによってはベーシックの中にこの文字関数がふくまれているものもある。

マシン語の世界はこの「16進法」の方が主流だし，ほとんどのパソコンのキャラクターが横8ビット単位で構成されているし，覚えておいて損のない数え方だ。

作戦資料105　中間言語と格納法

ホップ編で少しふれことに数字の大きさに制限があったね。このファミリーベーシックでは，数をあつかうのに2バイト使っているんだ。1バイトで表わせる数はØから255の256通りだったね。2バイトなら

256×256＝65536

つまり，Øから65535までの65535通り，という制限はこの、一つの数を表わすのに2バイトを使う，という所からきていたのがわかるね。

㊟　パソコンでは，小数点も使える単精度実数（一つの数を4バイト使って覚える）。もっと大きく，かつ細かな数を使う倍精度（一つの数に8バイト）などの型があり，ファミコンのように一つの数に2バイト使うものを「整数型」という。

では，命令や関数はどうなっているんだろう。一番簡単な方法は，プログラムを開始する所から，打ち込んだ通りの文字で記録さす方法だ。たとえば

1Ø　LOCATE　5，12

のようにあると，最初の番地（Xとする）から順番に一文字ごとにアスキーコード（キャラクタ・テーブルBのコードナンバー）を書き込むわけだ。(図1)

この方法だと，上の1行に14バイトかかるのがわかる。

ところで，ファミコンベーシックには，命令や関数・記号など，全部あわせても百ちょっとの種類しかない。そこで，一つの命令や記号：関数に一つづつ対応する数字をあてはめたらどうだろう。たとえば「LOCATE」なら「&HAE＝174」というふうに。ただ，こうすると，その1バイトが，命令用の数字なのか，定数用の数字なのかわからなくなる。

そこで，ファミコンでは「&H12＝18」という数字があると，次の2バイトは必ず数字が入っていると決められている。これを表わしたのが（図2）だ。

㊟　Z—8Ø」やファミコンに入っている「65Ø2」というCPUを使った場合，2バイトで数字を表わす時，

3A，6Ø

と並んでいると

&H6Ø×256＋&H3A

という数値になる。FM—7やS—1に使われている「68Ø9」というCPUでは素直に

&H3A×256＋&H6Ø

という意味なんだ。

なんとも，めんどうくさいね。

●図1

		キャラクター
X番地		1
X＋1		Ø
2	32	空白
3		L
4		O
5		C
6		A
7		T
8		E
9	32	空白
1Ø		5
11	44	，
12		1
13		2

●図2

		キャラクター
	1Ø	
1	Ø	
2	32	空白
3	174	LOCATE
4	32	空白
5	18	
6	Ø5	
7	ØØ	
8	44	，
9	18	
1Ø	12	
11	Ø	

（10進法表示）

（図1）にくらべて2バイト少なくてすむ。ほんの少ししか縮まらないようでも，ちりもつもればで，長いプログラムになるとバカにならない。

この方式にはもう一つ利点がある。コンピューターは普通，一回に1バイトしか見ることができない。とすると，(図1)の方式では。「LOCATE」という命令を実行するには，6バイトの中を見ていかなければならない。ところが，(図2)の方式では，わずか1バイトで，「LOCATE」命令だとわかる。プログラムの実行速度が，かなり違ってくるんだ。この(図2)の方式を「中間言語方式」という。つまり，人間の使った命令の単語をコンピューターのわかるマシン語との中間の言葉に訳して記録するわけなんだ。実際にはこの（図2）の一つ手前「X—1」番地の1バイトも大事な役割があるんだ。「X—1」番地にはこの行の長さを書いた数値が入るんだ。そして，1行の終わりには必ず「ØØ」が入っている。(図3)

そして「ØØ」の次にまた行の長さを表わす数値，行番号と続いているんだ。この行の長さを表わす数値をポインターと呼ぼう。Y番地がポインターなら，次のポインターは

Y＋（Y番地に書いてある数値）

で計算される番地にあるわけだ。

そして，このポインターは常に1バイト，行番号は2バイト，また，行番号と最初の命令の間は必ず空白が1個入るという約束のため，実際には記録されない。

1Ø LOCATE4，3

と入力してもLISTをとると

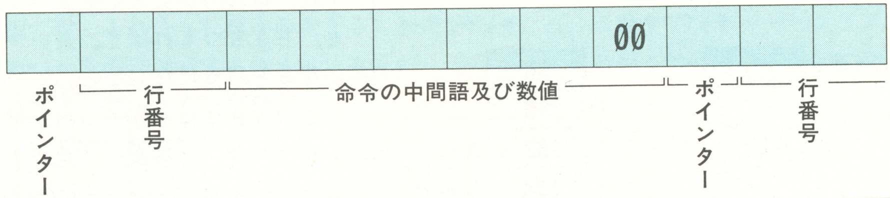
1Ø　LOCATE4，3

となることからもわかるだろう。

ホップ編でふれた二つの約束事も，これでわかるだろう。行番号がØ～65535までなのは，行番号を表わすのに2バイトしか使わない，という約束のせいなんだ。

1行の長さが255というのもポインターが数えられる1行の長さが1バイト以下という制

●図3



約のせいなんだ。

そして，この255という数値が，プログラムの命令の語数や，文字数でない。というのは中間言語で格納されている，その長さ，という意味なんだ。

ステップ編でもふれたVer. 3で使ったプログラムをVer. 2で読み込むとおかしくなる。というのもここの問題だ。

一つの数値を表わす時に，3バイト使うというのがファミコンの決まりなのに，メモリー節約のため，Ver. 3ではその数値が（10進法）で1ケタの時だけ，1バイトで表わす。というやり方をしている。ところが，そのやり方でTAPEへSAVEしたものをVer. 2で読むと，Ver. 2の方ではそんな読み方をしないからおかしくなる。というわけだ。

これをふせぐためにはVer.3でプログラムを書く時に，1ケタの数がでてきたら，必ず「&H」を頭につけるといい。

16進法でのØ～9は，16進法でも同じくØ～9と書かれる。そして，Ver.3でも16進法で書いてある時は，1ケタでもVer. 2と同じく3バイト使っているからなんだ。（最初は1バイトが&H11＝17　だと，次の2バイトが16進法で書いた数値という意味だ）

なんともややこしい話だね。

（図4）に中間言語のコード表をのせる。これをのぞくには，「NEW」したあと何かの短かいプログラムをうち込んで，ダイレクトに次の命令を行うといい。

```
FOR I=XX TO XX+3Ø:PRINT
HEX$(PEEK (I)) ; ";" ;:NEXT
  XXはVer.2では &H7Ø3F
      Ver.3では &H6ØØ6
```

●図 4

中間言語コード表(Ver.2+Ver.3)

	8Ø	9Ø	AØ	BØ	CØ	DØ	EØ	FØ
Ø	GOTO	LINPUT	VIEW	ERA	CAN	RND	LEFT$	OR
1	GOSUB	DATA	MOVE	TR		SGN	SCR$	AND
2	RUN	IF	END	FIND		SPC	INSTR	NOT
3	RETURN	READ	PLAY	GAME		TAB	CRASH	< >
4	RESTORE	DIM	BEEP	BGTOOL		MID$	ERR	> =
5	THEN	REM	LODE	AUTO		STICK	ERL	< =
6	LIST	STOP	SAVE	DELETE		STRIG	VCT	=
7	SYSTEM	CONT	POSITION	RENUM		XPOS		>
8	TO	CLS	KEY	FILTER		YPOS		<
9	STEP	CLEAR	COLOR	CLICK		VAL		+
A	SPRITE	ON	DEF	SCREEN	ABS	POS		−
B	PRINT	OFF	CGEN	BUCKUP	ASC	CSRLIN		MOD
C	FOR	CUT	SWAP	ERROR	STR$	CHR$		／
D	NEXT	NEW	CALL	REJUM	FRE	HEX$		*
E	PAUSE	POKE	LOCATE	BGPUT	LEN	INKEY$		
F	INPUT	CGSET	PALET	BGGET	PEEK	RIGHT$	XOR	

作戦資料106 マシン語とBASIC

コンピューターは,何回も書いた通り,人間の言葉を理解できない。わかるのは2進法数字で表わせるON・OFFばかりの情報だ。

何十万,何百万も並んだ「Ø」と「1」だけの情報の組み合わせで,ゲームからロケットを飛ばしたりまでするんだからえらいもんだ。と感心してばかりはいられない,この「Ø」と「1」との組み合わせを4個ごとに区切って1ケタの数字で表わしたものを16進法というのは,もう説明したよね。

Ø1ØØ1Ø1Ø1111Ø1Ø1

というのを

4AF5

と書くと少しはわかりやすい。でも「4AF5」と言われても,やはり人間には何のことだか,

ちっともわからない。

この数字で表わされたものを「マシン語」という。一つの数値（普通16進法で１ケタまたは２ケタ）について，コンピュータは一つの働きしかしない。そこで，一つの数値について人間のわかりやすそうな一つの単語をあてはめよう。(ベーシックの中間言語の考え方と逆だね）そして，プログラムを作る時にはこの単語を作って，それを自動的に数値に変換しよう、と考えられたのが「アセンブラ」だ。

ファミコンは「65Ø2」というCPUを使っているので，たとえばマシン語サブルーチンからの復帰（BASICでいうRETURN）は&H6Øにたいして，「RTS＝Return　Subroutin」という単語をあてはめるわけだ。

この「RTS」のような単語（ニーモックの）や数値（ニーモックの中で計算等をする命令—オペコード—の計算の対象になるもの＝オペランド）を書いたプログラムを「ソース・プログラム」といい，これを数値（マシン語）に翻訳するプログラム「アセンブラ」によって，マシン語に書き変えられたものを「オブジェクト・プログラム」という。

逆に，マシン語で書かれた（数値ばかりの）ものを，単語（ニーモック）に直すことを「逆アセンブル」といい，そのためのプログラムを「逆アセンブラ」という。

この，翻訳のための「アセンブラ」や「逆アセンブラ」はなくてもプログラムは組める。どの数値がどういう働きをするかを暗記しとけばいいわけで，これらの翻訳プログラムなしに，アセンブルすることを，手でするアセンブル，「ハンド・アセンブル」ともいうが，これのできる人はめったにいない。

だが「RTS」や「ADC」や「DEX」なんて単語だって覚えるのが大変だ。もっと人間にわかりやすいプログラムの書き方があるはずだ。

ということで，できあがったのが，「高級言語」といわれるものだ。

たとえば画面に一文字出力させるためのマシン語は，何十バイトもいる。それをいちいちプログラムするのはめんどうだ。あるまとまった動作を一つの単語で表わして，その単語が出たらそのひとまとまりの動作をさしてしまおう。というわけなんだ。

「アセンブラ」がCPUの一動作につき一つの単語を使うのにくらべ，「高級言語」の一単語は，一固まりの動作を表わしている。「高級言語」で「物を取る」という命令は「アセンブラ」では「息をする」「体をまげる」「手をのばす」「物をつかむ」というように，いくつもの命令になると考えてほしいんだ。

この「高級言語」にはいろんなものがある。技術計算に使うFORTRAN（フォートラン），事務用のCOBOL（コボル），そしてPASCAL（パスカル），LISP（リスプ），ADA（エイダ），今はやりのC（シー），そしてわれらがBASIC。

この高級言語にも二つの使い方がある。「コンパイラ」と「インタプリタ」だ。

「コンパイラ」というのは「アセンブラ」に近い。まず，何らかの高級言語で書かれたプログラムを翻訳係のプログラムがマシン語になおしてくれる。訳し終わった「オブジェクト・プログラム」を実行する時には，マシン語プログラムとして扱われる。つまり，プログラムを実行している時には，「コンパイラ」や「ソース・プログラム」はいらないわけだ。

この方式では，できあがったプログラムはマシン語だから，スピードは速い。ただし，もしプログラムに間違いがあっても教えてくれないし，ほんの少しの手直しにも非常に時間がかかる。

今，市販されているゲームのほとんどが，この「コンパイラ」や「アセンブラ」を使って書かれたマシン語のプログラムなのは，どうしてもリアルタイムゲームではスピードが要求されるからだ。

「インタープリタ」は，「コンパイラ」と違って，マシン語プログラムである「オブジェクト・プログラム」を作らない。「ソース・プログラム」だけである。

たとえば，ベーシックで「RUN」とする。「インタープリタ」が「ソース・プログラム」を読んで，それをマシン語に通訳する。そして，その動作をおわると次の命令を読みにいって，また通訳する。いわば，「同時通訳」なのだ。

当然，「インタプリタ」はそのたびに通訳するため，時間がかかる，「コンパイラ」にくらべ数倍から数十倍の時間がかかる。そのかわり，もし間違いがあっても，その場で「エラー」として教えてくれる機能もついているし，何よりその場で修正したり，あるいはプログラムが途中までしかできなくてもテスト・ランもできる。

現在の高級言語はほとんど「コンパイラ」だけになってしまってって，「インタープリタ」はBASICぐらいしかない。もちろん「ベーシック・コンパイラ」もある。が，プロのプログラマー？達はBASICを悪く言う人も多いんだ。

しかし，人間用の言語としてはベーシックはやはり，最高だと思うのは私だけではないと思う。

Biginners All-purpose Symbolic
Instruction Code

というのがBASICの本当の意味だ。

普通，「ベーシックのプログラム」というのは，「BASIC」という規格（きまり）を使って書いたプログラム，という意味で，単に「BASIC」という時は，それ以外に，ベーシックの同時通訳係＝「ベーシック・インタープリタ」または，ベーシックの翻訳係「ベーシック・コンパイラ」や「BASIC言葉」そのものを意味することもあるのを覚えておこう。

作戦資料107 ファミコンとパソコン—LIST OUT

ファミコン・ゲーム集でも，またこの本でもファミコンのプログラムをリストアウトしている。でも，ファミコンにはプリンターをつなげないはずだ。それに，ファミコンの「バ」が「ダ」が「ハ゛」，「タ゛」と二文字になっている。

実はこのプログラム・リストは，ファミコンで作ったプログラムをSAVEしたテープを，「FM—7」。というパソコンでLOADし，プリントアウトしたんだ。

「エッ？　FM—7でファミコンのテープを読めるか？」もちろん普通のままでは読めない。テープに録音する速さ（ボーレートという）も違うし，データの記入法も違う。ちょっとしたプログラムが必要だ。

でも，一回打ち込むとFM—7さえあればファミコンのプログラムをリストアウトできる。デバックの時，画面上にリストを表示するより，すでに印刷されたものを見直すのが，非常に有効なのは間違いない。

まず，FM—7上でダイレクトに

CLEAR, &H4FFF

と入力したあと，マシン語プログラム（リスト1）を打ち込んでほしい。打ち終わると，

SAVE"LOADER-M", &H529Ø, &H5BFF, &H529Ø

とディスクやカセットにSAVEしよう。

次に，BASICプログラム（リスト2）を打ち込んで，ディスクやカセットにSAVEする。

実際に動作するには，ベーシックプログラムを「RUN」すると自動的にマシン語を「LOAD」する（テープではリスト2，リスト1の順にSAVEしてほしい）。

入力ミスのないかぎり，「RUN」すると

@

のマークが出るので，ファミコンのテープをカセットに入れ「PLAY」ボタンを押す（リモート端子コードは抜くこと）。「ピー」という音が聞こえたらRETURNキーを押そう（他のキーを押すとベーシックにもどる）。

画面にベーシックかBG画面かの表示と，ファイル名が表示される。LOADが終わると，BGデーターの場合はすぐにプリントアウトされる。

ベーシックの場合は

HIT　PF—1Ø

と表示されるので，ファンクションキーの10番を押す。これでLISTを取るとみごとにファミコンのプログラムがFM—7上にできているはずだ。このプログラムをFM—7上で，ディスクやテープにSAVEしておくと，いつまでも保存できるしくみだ。

ただファミコンの場合，ファミコンやテープレコーダーへSAVEする時，レコーダーによって音が違うことが多い。

ファミコン──→テープ	NEC	PC—6Ø82
テープ──→FM—7	SANYO	MR—11—DR
プリンター	EPSON	RP—8Ø

というのが今回使った機器だった。(サンヨーのMR—11—DRは富士通の純正データレコーダーと全く同じ)

BG画面は横28字で，各字につき3文字＋わくが入るので普通の8Øケタプリンターでは横に表示できない。170行は8Øケタプリンターである RP—8Øで縮少文字モードにするための命令だ。他のプリンターを使う時はそれぞれの縮小文字モードにするコードに変更してほしい。マシン語プログラムリスト1のうち&H529Øから&H52DFがテープを読み込むためのプログラム。そのあとは中間言語からベーシックの命令へ変更するためのテーブルやワークだ。

テープからLOADした直後のデータは&H6ØØØから入っている。ベーシック・プログラムをLOADして，ファミコンのTV画面でLISTをとったものとくらべて見ると，ベーシックがどんな形で格納されているかよくわかる。(リストがどんな形で入っていたかを**リスト3**に示す―この本で作ったロールプレイング・ゲームのリストの先頭部分。□で囲った2バイトが行番号の部分だ)

BASIC	ヘッダー	&H6ØØØ～
	プログラム	&H61ØØ～
BGデーター	ヘッダー	&H6ØØØ～
	データⅠ	&H61ØØ～
	ヘッダー	&H62ØØ～
	データⅡ	&H63ØØ～
	ヘッダー	&H64ØØ～
	データⅢ	&H65ØØ～
	ヘッダー	&H66ØØ～
	データⅣ	&H67ØØ～&H67FF

BG画面をなんでこんなに四つにわけたか，意味はよくわからない。

なお，このプログラムはFM—7，77，NEW7のディスク，テープ両方で使える。

ファミコンのキャラクターでFM—7にないものは変換してない。また「ファ」は「ファ」，「©」は「○」，「÷」は「／」になおした。

またVer. 2でSAVEしたものでもVer. 3でSAVEしたものでも大丈夫だ。(1ケタの数字に&Hを使わなくてもよい→バグ情報ステップ参照)

●リスト1

```
5290     20 0C 00 00 00 00 00 00 00 00 00 00 00 00 1A 50
52A0     86 40 BD D9 D9 BD DB 6D 27 FB 81 0D 27 02 20 5D
52B0     86 41 B7 FD 03 10 FF 52 93 7F 52 92 86 04 B7 52
52C0     97 8E 60 00 BF 52 98 8E 00 80 BF 52 9A 86 28 B7
52D0     52 9C 8D 45 B6 52 97 81 04 26 03 17 00 A8 BE 52
52E0     98 E6 88 11 A6 88 12 FD 52 9A 30 89 01 00 BF 52
52F0     98 86 14 B7 52 9C 8D 21 B6 52 92 81 02 27 0E 7A
5300     52 97 27 09 BE 52 98 30 89 01 00 20 B7 1C AF B6
5310     52 92 81 02 10 26 00 C8 39 FE 52 98 BE 52 9A 5F
5320     8D 3F 2A F5 5C F1 52 9C 25 F6 5F 8D 34 2B F0 5C
5330     F1 52 9C 25 F6 8D 2A B6 52 9C 81 14 27 0A 8D 11
5340     7D 52 92 26 03 F7 52 92 8D 07 E7 C0 30 1F 26 F8
5350     39 8D 0E C6 08 F7 52 96 8D 07 59 7A 52 96 26 F8
5360     39 8D 0F 2A FC 7F 52 95 8D 08 2B FC 86 14 B0 52
5370     95 39 7C 52 95 B6 FD 04 84 02 26 06 10 FE 52 93
5380     20 8B B6 FD 02 39 B6 52 92 81 02 26 05 8E 53 BB
5390     20 03 8E 53 C7 A6 80 2B 05 BD D9 D9 20 F7 8E 60
53A0     00 CE 56 40 A6 80 26 01 39 C6 02 3D EC CB BD D9
53B0     D9 5D 27 F0 1F 98 BD D9 D9 20 E9 20 42 61 73 69
53C0     63 20 2D 2D 2D 20 80 20 42 47 20 67 72 61 70 68
53D0     69 63 20 2D 2D 2D 20 80 00 00 00 00 00 00 00 00
53E0     20 0B 00 00 00 00 00 00 00 00 00 00 00 8E 61 00
53F0     10 8E 5C 00 33 89 01 00 FF 53 E2 20 18 30 89 01
5400     00 8C 68 C0 27 34 8C 67 00 26 E9 CE 67 C0 20 E8
5410     BC 53 E2 27 E8 A6 84 C6 02 3D CE 56 40 EE CB EF
5420     A1 A6 80 81 20 27 0A 25 0B 81 B8 24 07 86 01 20
5430     05 4F 20 02 86 FF A7 A0 20 D6 10 8E 67 C0 CE 5C
5440     02 86 08 B7 53 EC 8E 53 E4 C6 08 A6 A0 A7 80 5A
5450     26 F9 CC 02 08 FD 53 E2 8E 53 E4 8D 21 8D 1F 30
5460     01 7A 53 E3 26 F5 33 C8 60 7A 53 E2 27 05 CC 01
5470     08 20 E2 7A 53 EC 27 02 20 CC FF 53 E2 39 A6 84
5480     84 03 64 84 64 84 8B 30 8D 03 8D 01 39 5F 8D 07
5490     C6 60 8D 03 33 43 39 34 12 A6 C5 27 0C 81 01 26
54A0     0C C0 02 AE C5 5C AF C5 5A 86 20 20 02 A6 E4 A7
54B0     C5 35 12 39 00 00 00 00 00 00 00 00 00 00 00 00
54C0     20 08 00 00 00 00 00 00 00 00 8E 00 33 EE 84 20
```

```
54D0    08 CC 00 00 ED C1 DF 35 39 8E 60 FF 6F 84 EC 84
54E0    27 EF 8D 39 7F 54 C9 7D 54 C9 26 12 A6 80 2A 10
54F0    81 95 26 05 C6 02 F7 54 C9 17 03 44 20 04 A6 80
5500    8D 3E BC 54 C4 2A 09 A6 1F 81 3A 27 DA 4F 20 D7
5510    27 07 86 41 B7 FD 03 20 F9 8D 1D 20 C1 10 AE 02
5520    FF 54 C2 33 42 34 06 1F 20 1E 89 ED C1 35 06 31
5530    8B 10 BF 54 C4 30 04 39 4F A7 C0 EF 9F 54 C2 39
5540    10 27 00 89 81 0B 27 02 81 12 26 53 E6 80 A6 80
5550    7F 54 C8 10 8E 27 10 8D 17 10 8E 03 E8 8D 11 10
5560    8E 00 64 8D 0B 10 8E 00 0A 8D 05 CB 30 E7 C0 39
5570    6F C4 10 BF 54 C6 10 B3 54 C6 2B 07 B3 54 C6 6C
5580    C4 20 F3 6D C4 26 05 7D 54 C8 27 0D 34 02 A6 C4
5590    8B 30 A7 C0 35 02 7C 54 C8 39 1F 89 5A 20 CC 81
55A0    0A 25 F7 81 11 26 22 CC 26 48 ED C1 E6 80 A6 80
55B0    7F 54 C8 34 04 1F 89 8D 26 35 02 1F 89 8D 20 7D
55C0    54 C8 26 04 86 30 A7 C0 39 81 20 24 35 C6 01 F7
55D0    54 C8 C6 9C E7 C0 1F 89 8D 05 C6 9D E7 C0 39 44
55E0    44 44 44 8D 07 1F 98 84 0F 8D 01 39 4D 26 05 7D
55F0    54 C8 27 0D 8B 30 81 3A 2B 02 8B 07 A7 C0 7C 54
5600    C8 39 81 B8 24 C7 81 22 26 13 F6 54 C9 27 09 C1
5610    02 27 18 7F 54 C9 20 13 7C 54 C9 20 0E 81 27 26
5620    0A F6 54 C9 26 05 C6 02 F7 54 C9 C6 02 3D 10 8E
5630    56 40 EC AB A7 C0 C1 20 27 02 E7 C0 39 00 00 00
5640    41 30 41 31 41 32 41 33 41 34 41 35 41 36 41 37
5650    42 30 42 31 42 32 42 33 42 34 42 35 42 36 42 37
5660    43 30 43 31 43 32 43 33 43 34 43 35 43 36 43 37
5670    44 30 44 31 44 32 44 33 44 34 44 35 44 36 44 37
5680    20 20 21 20 22 20 23 20 24 20 25 20 26 20 27 20
5690    28 20 29 20 2A 20 2B 20 2C 20 2D 20 2E 20 2F FE
56A0    C0 FE C0 05 FF FE C0 F8 BE F8 BE F8 BE F6 22 05
56B0    A0 F1 9A F8 BE F6 22 05 A0 F8 BE F6 22 D4 B0 F1
56C0    9A D4 B0 A8 05 00 00 D4 B0 C4 DD 07 1E 8F 5D C6
56D0    7E 00 49 20 4A 20 4B 20 4C 20 4D 20 4E 20 4F 20
56E0    50 20 51 20 52 20 53 20 54 20 55 20 56 20 57 20
56F0    58 20 59 20 5A 20 A2 20 5C 20 A3 20 5E 20 B0 20
5700    B1 20 B2 20 B3 20 B4 20 B5 20 B6 20 B7 20 B8 20
5710    B9 20 BA 20 BB 20 BC 20 BD 20 BE 20 BF 20 C0 20
5720    C1 20 C2 20 C3 20 C4 20 C5 20 C6 20 C7 20 C8 20
5730    C9 20 CA 20 CB 20 CC 20 CD 20 CE 20 CF 20 D0 20
```

```
5740    D1 20 D2 20 D3 20 D4 20 D5 20 D6 20 D7 20 D8 20
5750    D9 20 DA 20 DB 20 DC 20 DD 20 A6 20 A7 20 A8 20
5760    A9 20 AA 20 AB 20 AC 20 AD 20 AE 20 AF 20 B6 DE
5770    B7 DE B8 DE B9 DE BA DE BB DE BC DE BD DE BE DE
5780    BF DE C0 DE C1 DE C2 DE C3 DE C4 DE CA DE CB DE
5790    CC DE CD DE CE DE CA DF CB DF CC DF CD DF CE DF
57A0    F4 20 A1 20 5B 20 5D 20 ED 20 F0 20 90 20 CC A7
57B0    45 30 45 31 45 32 45 33 45 34 45 35 45 36 45 37
57C0    46 30 46 31 46 32 46 33 46 34 46 35 46 36 46 37
57D0    47 30 47 31 47 32 47 33 47 34 47 35 47 36 47 37
57E0    48 30 48 31 48 32 48 33 48 34 48 35 48 36 48 37
57F0    49 30 49 31 49 32 49 33 49 44 55 4D 50 00 04 37
5800    4A 30 4A 31 4A 32 4A 33 4A 34 4A 35 4A 36 4A 37
5810    4B 30 4B 31 4B 32 4B 33 4B 34 4B 35 4B 36 4B 37
5820    4C 30 4C 31 4C 32 4C 33 4C 34 4C 35 4C 36 4C 37
5830    4D 30 4D 31 4D 32 4D 33 4D 34 4D 35 4D 36 4D 37
5840    10 8E 58 59 80 80 C6 02 3D EC AB 83 04 00 1F 02
5850    A6 A0 2B 04 A7 C0 20 F8 39 5D 60 5D 65 5D 6B 5D
5860    6F 5D 76 5D 7E 5D 83 5D 88 5D 8F 5D 92 5D 97 5D
5870    9E 5D A4 5D A8 5D AD 5D B3 5D B9 5D C0 5D C5 5D
5880    C8 5D CD 5D D1 5D D5 5D DA 5D DF 5D E3 5D E9 5D
5890    EC 5D F0 5D F4 5D F8 5D FD 5E 03 5E 08 5E 0D 5E
58A0    11 5E 16 5E 1B 5E 20 5E 25 5E 2E 5E 32 5E 38 5E
58B0    3C 5E 41 5F EE 5E 4A 5E 51 5E 57 5E 5B 5E 5E 5E
58C0    63 5E 68 5E 6F 5E 74 5E 7B 5E 81 5E 88 5E 8E 5E
58D0    95 5E 9C 5E A2 5E A8 5E AE 5E B4 5E B8 5E BF 5E
58E0    C6 5E CD 5E D4 5E DB 5E E2 5E E9 5E F0 5E F7 5E
58F0    FB 5E FF 5F 04 5F 08 5F 0C 5F 11 5F 15 5F 19 5F
5900    1D 5F E7 5F 25 5F 2B 5F 31 5F 36 5F 3B 5F 3F 5F
5910    43 5F 4A 5F 4F 5F 54 5F 5B 5F 62 5F 68 5F 6D 5F
5920    73 5F 79 5F 7D 5F 81 5F 85 5F 8C 5F 93 5F 9A 5F
5930    A1 5F A8 5F AF 5F B6 5F BD 5F C1 5F C4 5F C8 5F
5940    CC 5F CF 5F D2 5F D5 5F D7 5F D9 5F DB 5F DD 5F
5950    DF 5F E3 5F E5 5F E7 5F EE 5F F5 00 00 00 00 00
5960    47 4F 54 4F 81 47 4F 53 55 42 81 52 55 4E 81 52
5970    45 54 55 52 4E 81 52 45 53 54 4F 52 45 81 54 48
5980    45 4E 81 4C 49 53 54 81 53 59 53 54 45 4D 81 54
5990    4F 81 53 54 45 50 81 53 50 52 49 54 45 81 50 52
59A0    49 4E 54 81 46 4F 52 81 4E 45 58 54 81 50 41 55
```

59B0	53	45	81	49	4E	50	55	54	81	4C	49	4E	50	55	54	81
59C0	44	41	54	41	81	49	46	81	52	45	41	44	81	44	49	4D
59D0	81	52	45	4D	81	53	54	4F	50	81	43	4F	4E	54	81	43
59E0	4C	53	81	43	4C	45	41	52	81	4F	4E	81	4F	46	46	81
59F0	43	55	54	81	4E	45	57	81	50	4F	4B	45	81	43	47	53
5A00	45	54	81	56	49	45	57	81	4D	4F	56	45	81	45	4E	44
5A10	81	50	4C	41	59	81	42	45	45	50	81	4C	4F	44	45	81
5A20	53	41	56	45	81	50	4F	53	49	54	49	4F	4E	81	4B	45
5A30	59	81	43	4F	4C	4F	52	81	44	45	46	81	43	47	45	4E
5A40	81	53	57	41	50	81	43	41	4C	4C	4C	4F	43	41	54	45
5A50	81	50	41	4C	45	54	81	45	52	41	81	54	52	81	46	49
5A60	4E	44	81	47	41	4D	45	81	42	47	54	4F	4F	4C	81	41
5A70	55	54	4F	81	44	45	4C	45	54	45	81	52	45	4E	55	4D
5A80	81	46	49	4C	54	45	52	81	43	4C	49	43	4B	81	53	43
5A90	52	45	45	4E	81	42	55	43	4B	55	50	81	45	52	52	4F
5AA0	52	81	52	45	53	55	4D	81	42	47	50	55	54	81	42	47
5AB0	47	45	54	81	43	41	4E	81	2A	4E	4F	4F	50	2A	81	2A
5AC0	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F
5AD0	4F	50	2A	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50
5AE0	2A	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50	2A	81
5AF0	2A	4E	4F	4F	50	2A	81	41	42	53	81	41	53	43	81	53
5B00	54	52	24	81	46	52	45	81	4C	45	4E	81	50	45	45	4B
5B10	81	52	4E	44	81	53	47	4E	81	53	50	43	81	54	41	42
5B20	81	4D	49	44	24	53	54	49	43	4B	81	53	54	52	49	47
5B30	81	58	50	4F	53	81	59	50	4F	53	81	56	41	4C	81	50
5B40	4F	53	81	43	53	52	4C	49	4E	81	43	48	52	24	81	48
5B50	45	58	24	81	49	4E	4B	45	59	24	81	52	49	47	48	54
5B60	24	81	4C	45	46	54	24	81	53	43	52	24	81	49	4E	53
5B70	54	52	81	43	52	41	53	48	81	45	52	52	81	45	52	4C
5B80	81	56	43	54	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F	4F
5B90	50	2A	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50	2A
5BA0	81	2A	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50	2A	81	2A
5BB0	4E	4F	4F	50	2A	81	2A	4E	4F	4F	50	2A	81	58	4F	52
5BC0	81	4F	52	81	41	4E	44	81	4E	4F	54	81	3C	3E	81	3E
5BD0	3D	81	3C	3D	81	3D	81	3E	81	3C	81	2B	81	2D	81	4D
5BE0	4F	44	81	2F	81	2A	81	4D	49	44	24	81	2A	81	43	41
5BF0	4C	4C	81	2A	81	FF	FF	00	00	00	00	00	00	00	00	00

●リスト2

```
10 KEY 10,"EXEC&H54C0"+CHR$(13)
20 CLEAR ,&H4FFF
40 LOADM"B-RDFUM":'TOTAL
50 POKE &H5292,0
100 EXEC &H5290
110 FRUG=PEEK(&H5292)
120 IF FRUG=3 THEN 160
125 IF FRUG<>2 THEN 140
130 PRINT:PRINT "HIT   PF-10"
140 END
160 ADR=&H5C00+32*3*3+6
170 LPRINT CHR$(15);CHR$(&H1B);"A";CHR$(5);
180 T$="  ":FOR I=0 TO 27:T$=T$+RIGHT$(("  "+STR$(
I)+" "),4):NEXT I:LPRINT T$
190 T$="  ┌───":FOR I=0 TO 26:T$=T$+ "┬───":NEXT I
:T$=T$+"┐":LPRINT T$
200 FOR I=0 TO 20:T$=""+RIGHT$((" "+STR$(I)),2)
210 FOR J=0 TO 27:T$=T$+"|"
220 FOR K=1 TO 3:D$=CHR$(PEEK(ADR)):T$=T$+D$:ADR=A
DR+1:NEXT K
230 NEXT J:T$=T$+"|":ADR=ADR+12:LPRINT T$
240 IF I=20 THEN 260 ELSE T$="  ├───":FOR L=0 TO 2
6:T$=T$+ "┼───":NEXT L:T$=T$+"┤":LPRINT T$
250 NEXT I
260 T$="  └───":FOR I=0 TO 26:T$=T$+ "┴───":NEXT:T
$=T$+"┘":LPRINT T$
270 LPRINT CHR$(&H12);CHR$(&H1B);"0";:END
```

●リスト3

```
6100    18 0A 00 98 3A 94 41 28 12 07 00 29 2C 42 28 12
6110    02 00 2C 12 01 00 29 00 3A 14 00 8C 49 F6 12 01
6120    00 88 12 04 00 3A 41 28 49 29 F6 12 0F 00 3A 8D
6130    3A 41 28 12 05 00 29 F6 12 01 00 3A 41 28 12 06
6140    00 29 F6 12 01 00 3A 41 28 12 07 00 29 F6 12 0B
6150    00 00 2E 1E 00 8C 49 F6 12 00 00 88 12 02 00 3A
6160    8C 4A F6 12 00 00 88 12 01 00 3A 42 28 49 2C 4A
6170    29 F6 12 0A 00 F9 49 FD 12 05 00 3A 8D 3A 8D 00
6180    20 28 00 4F F6 D0 28 12 14 00 29 F9 12 50 00 3A
6190    92 4F 20 FB 12 0A 00 F6 12 00 00 85 0B 28 00 00
61A0    1D 32 00 42 24 F6 22 7C 61 6B 20 9B 2D 89 80 A5
61B0    86 9B 8C 85 62 6D 61 69 A7 73 20 22 00 1B 3C 00
61C0    41 24 F6 22 6B 95 67 87 95 62 A8 66 89 A9 88 20
61D0    67 6C 87 20 20 20 22 00 6E 64 00 8E 12 32 00 3A
61E0    8B 22 9A 8C 9C 61 20 22 3B DC 28 12 40 00 F9 41
61F0    28 12 07 00 29 FC 12 0A 00 29 3B 22 2D 22 3B 41
6200    28 12 07 00 29 FB 12 0A 00 3A 8B 22 68 61 68 8C
6210    70 3D 22 3B 41 28 12 00 00 29 3A 8B 22 6F 61 87
6220    95 67 3D 22 3B 41 28 12 01 00 29 3A 8B 22 6D 8C
6230    73 62 87 95 67 3D 22 3B 41 28 12 02 00 29 FD 41
6240    28 12 05 00 29 00 0D C8 00 8B 22 69 7E 8C A5 79
6250    3F 22 00 15 D2 00 41 F6 CB 28 DE 29 3A 92 41 F6
6260    12 00 00 85 0B D2 00 00 14 DC 00 92 41 F8 12 20
6270    00 F1 41 F7 12 1B 00 85 0B A0 0F 00 0E E6 00 92
6280    41 F6 12 49 00 85 0B E8 03 00 0E F0 00 92 41 F6
6290    12 45 00 85 0B D0 07 00 0E FA 00 92 41 F6 12 44
62A0    00 85 0B B8 0B 00 08 04 01 80 0B D2 00 00 37 E8
62B0    03 8C 49 F6 12 00 00 88 12 02 00 3A 8B D4 28 41
62C0    24 2C 49 FD 12 06 00 F9 12 01 00 2C 12 06 00 29
62D0    3B 22 3D 22 3B 41 28 49 F9 12 04 00 29 3A 8D 3A
62E0    80 0B 64 00 00 13 D0 07 92 41 28 12 04 00 29 F8
62F0    12 01 00 85 0B C4 09 00 2C DA 07 41 28 12 01 00
6300    29 F6 41 28 12 01 00 29 F9 12 02 00 3A 41 28 12
6310    04 00 29 F6 41 28 12 04 00 29 FA 12 01 00 3A 80
6320    0B AC 0D 00 13 C4 09 A4 3A 8B 22 A4 66 74 61 21
6330    22 3A 80 0B 64 00 00 13 B8 0B 92 41 28 12 06 00
6340    29 F8 12 01 00 85 0B C4 09 00 27 C2 0B 41 28 12
6350    01 00 29 F6 41 28 12 01 00 29 F9 12 04 00 3A 41
6360    28 12 06 00 29 F6 41 28 12 06 00 29 FA 12 01 00
6370    00 20 AC 0D 92 41 28 12 01 00 29 F7 41 28 12 03
6380    00 29 85 41 28 12 01 00 29 F6 41 28 12 03 00 29
6390    00 08 B6 0D 80 0B 88 13 00 6B A0 0F 42 F6 41 28
```

エラーメッセージ一覧表

エラーコード	エラー表示	エラーメッセージ	説明
Ø	NF	NEXT without FOR	FORがないのにNEXTがある。
1	SN	Syntax error	文法がまちがっている。
2	RG	RETURN without GOSUB	GOSUBがないのにRETURNがある。
3	OD	Out of DATA	READで読むべきデータがDATA文に用意されていない。
4	IL	Illegal function call	ステートメントや関数の呼び方がまちがっている。
5	OV	OverfloW	演算結果が許容範囲を越えた。
6	OM	Out of memory	メモリ不足である。
7	UL	Undefined line Number	GOTO, GOSUB, IFなどで指定した分岐先の行番号がない。
8	SO	Subscript out of range	配列変数の添字が規定外である。
9	DD	Duplicate Definition	配列が２重に定義されている。
10	DZ	Division by zero	Øによる除算をした。
11	TM	Type mismatch	変数の型が一致しない。
12	ST	String too long	文字が31文字を越えている。
13	FT	Formula Too complex	式が複雑すぎる。たとえば，（　）が異常に多い場合。
14	CC	Can't continue	CONTによってプログラムの実行を再開できない。
15	UF	Undefind function	未定義の関数を呼んだ。
16	MO	Missing operand	パラメータの必要な命令に指定がない。
17	TP	Tape read ERROR	カセットテープからデータが正しく読めない。
18	NR	No RESUME	ON ERROR GOTOで分岐先にRESUME文がない。
19	RE	RESUME without error	RESUME文のあるルーチンへON ERROR GOTO以外から分岐した。
20	NB	No BG data	ＢＧデータがRAM領域にない。
21	UP	Unprintable error	コードNO.Ø～NO.20までに含まれないエラー。

※コードNO.15，18～21はV3で新たに追加されたものです。

ゲームキャラクタ表(キャラクタテーブルA)



これらのキャラクタは、主にアニメキャラクタとして使用します。
各キャラクタの四隅に表記している数字は、DEF SPRITE文のCHR$(n)で使用する10進数です。

ゲームキャラクタ表(キャラクタテーブルB)



さくいん (A〜G)

H＝ホップ・S＝ステップ・J＝ジャンプ

さくいん (G~P)

H=ホップ・S=ステップ・J=ジャンプ

さくいん（S～Y）

H＝ホップ・S＝ステップ・J＝ジャンプ

検印省略　NDC　548

ファミリーベーシック ジャンプ

昭和60年7月24日　発行

定価　800円

著　者　地球防衛軍
発行者　小川茂男
発行所　㈱誠文堂新光社
東京都千代田区神田錦町1—5—5
郵便番号 101
電話　東京（292）1211
振替口座　東京　7—6294
印刷・広研印刷株式会社
製本・岡嶋製本工業

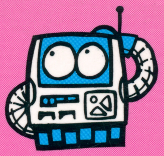
ISBN4-416-18543-X C2055

ファミコン大作戦

ファミリーベーシック™ ジャンプ

誠文堂新光社

ファミコン大作戦
ファミリーベーシック™ジャンプ



ファミコン大作戦

ファミコン大作戦 ファミリーベーシック™ジャンプ 誠文堂新光社

誠文堂新光社 ISBN4-416-18543-X C-2055 ¥800E 定価 800 円

ファミコン大作戦

ファミリーベーシック™

ジャンプ

誠文堂新光社

カラーチャート

スプライト用

パレットコード0

				配色番号
ⓐ	54 36	22 16	2 Ø2	0
ⓑ	39 27	48 3Ø	25 19	1
ⓒ	53 35	37 25	23 17	2
ⓓ	48 3Ø	39 27	22 16	3

パレットコード1

				配色番号
ⓔ	48 3Ø	22 16	1 Ø1	0
ⓕ	16 1Ø	Ø ØØ	1 Ø1	1
ⓖ	48 3Ø	41 29	9 Ø9	2
ⓗ	48 3Ø	22 16	7 Ø7	3

パレットコード2

				配色番号
ⓘ	48 3Ø	38 26	18 12	0
ⓙ	48 3Ø	21 15	18 12	1
ⓚ	48 3Ø	18 12	22 16	2
ⓛ	48 3Ø	38 26	25 19	3

バックグラウンド用

パレットコード0

				配色番号
ⓜ	44 2Ø	21 15	7 Ø7	0
ⓝ	39 27	33 21	18 12	1
ⓞ	41 29	54 36	23 17	2
ⓟ	48 3Ø	38 26	7 Ø7	3

パレットコード1

				配色番号
ⓠ	48 3Ø	33 21	2 Ø2	0
ⓡ	48 3Ø	39 27	24 18	1
ⓢ	48 3Ø	39 27	22 16	2
ⓣ	41 29	54 36	23 17	3